US005442305A

United States Patent [19]

Martin et al.

[11] Patent Number: 5,442,305

[45] Date of Patent: Aug. 15, 1995

[54] ACTIVE BUS TERMINATION DEVICE

[75] Inventors: **Stephen R. Martin**, San Jose; **Randall O. Mooney, Jr.**, Boulder Creek, both of Calif.

[73] Assignee: **Apple Computer, Inc.**, Cupertino, Calif.

[21] Appl. No.: **306,010**

[22] Filed: **Sep. 14, 1994**

Related U.S. Application Data

[62] Division of Ser. No. 771,127, Oct. 3, 1991, Pat. No. 5,357,519.

[51] **Int. Cl.**[6] **H03K 17/16**

[52] **U.S. Cl.** **326/30**; 326/31

[58] **Field of Search** 326/30-31, 326/38

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,450,370	5/1984	Davis	326/30
4,489,414	12/1984	Titherley	371/15.1 X
4,675,551	6/1987	Stevenson et al.	326/30
4,694,408	9/1987	Zaleski	364/551
4,703,482	10/1987	Auger et al.	371/16.1
4,748,426	5/1988	Stewart	326/30
4,760,329	7/1988	Andreano	371/15.1
4,804,861	2/1989	Hollstein et al.	326/30
4,810,958	3/1989	Mogi et al.	371/15.1 X
4,831,283	5/1989	Newton	326/30
4,831,560	5/1989	Zaleski	364/551.01
4,837,764	6/1989	Russello	371/16.1
4,953,165	8/1990	Jackson	371/16.1
5,029,284	7/1991	Feldbaumer et al.	326/30
5,111,080	5/1992	Mizukami et al.	326/30
5,157,665	10/1992	Fakraie-Fard et al.	371/15.1
5,168,216	12/1992	Dance	324/158 R
5,228,039	7/1993	Knoke et al.	371/19

OTHER PUBLICATIONS

Apple Computer, Inc., *Inside Macintosh. Vol. IV.* (1986), pp. IV-238 through IV-295.

Primary Examiner—David R. Hudspeth

Attorney, Agent, or Firm—Blakely, Sokoloff, Taylor & Zafman

[57] **ABSTRACT**

A diagnostic apparatus for testing devices such as computer systems, and computer system components such as disk drives or printers. The device comprises a main unit, the main unit having a central processing unit for executing instructions, issuing commands, and receiving data from a first device. The apparatus also has a first peripheral unit coupled to the main unit, the first peripheral unit having ports for interfacing with the first device, the first peripheral unit being interchangeable with a second peripheral unit for interfacing with a second device. The apparatus also comprises a first non-volatile memory unit coupled to the main unit. The first non-volatile memory unit comprising a first set of tests for the first device, the first non-volatile memory unit being interchangeable with a second non-volatile memory unit comprising a second set of tests for a second device. These interchangeable parts are provided so that the user may test various types of hardware. The apparatus further comprises software means for providing termination on a bus and methods for simulating devices on a bus for remote entry into diagnostic programs.

5 Claims, 17 Drawing Sheets

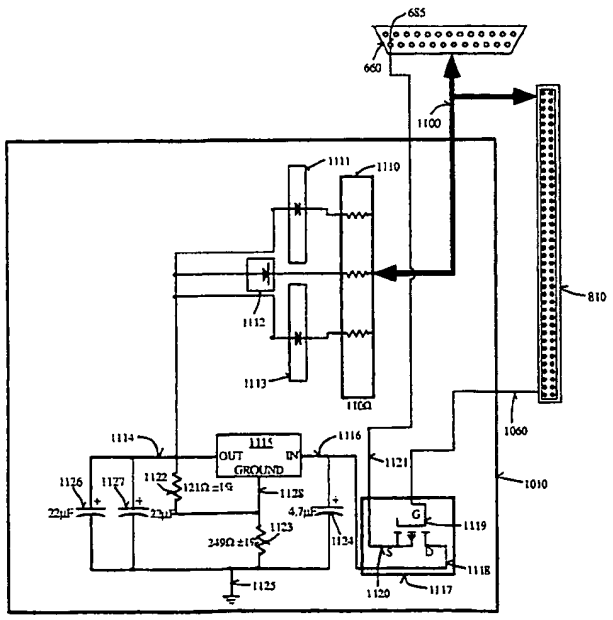


FIG. 1

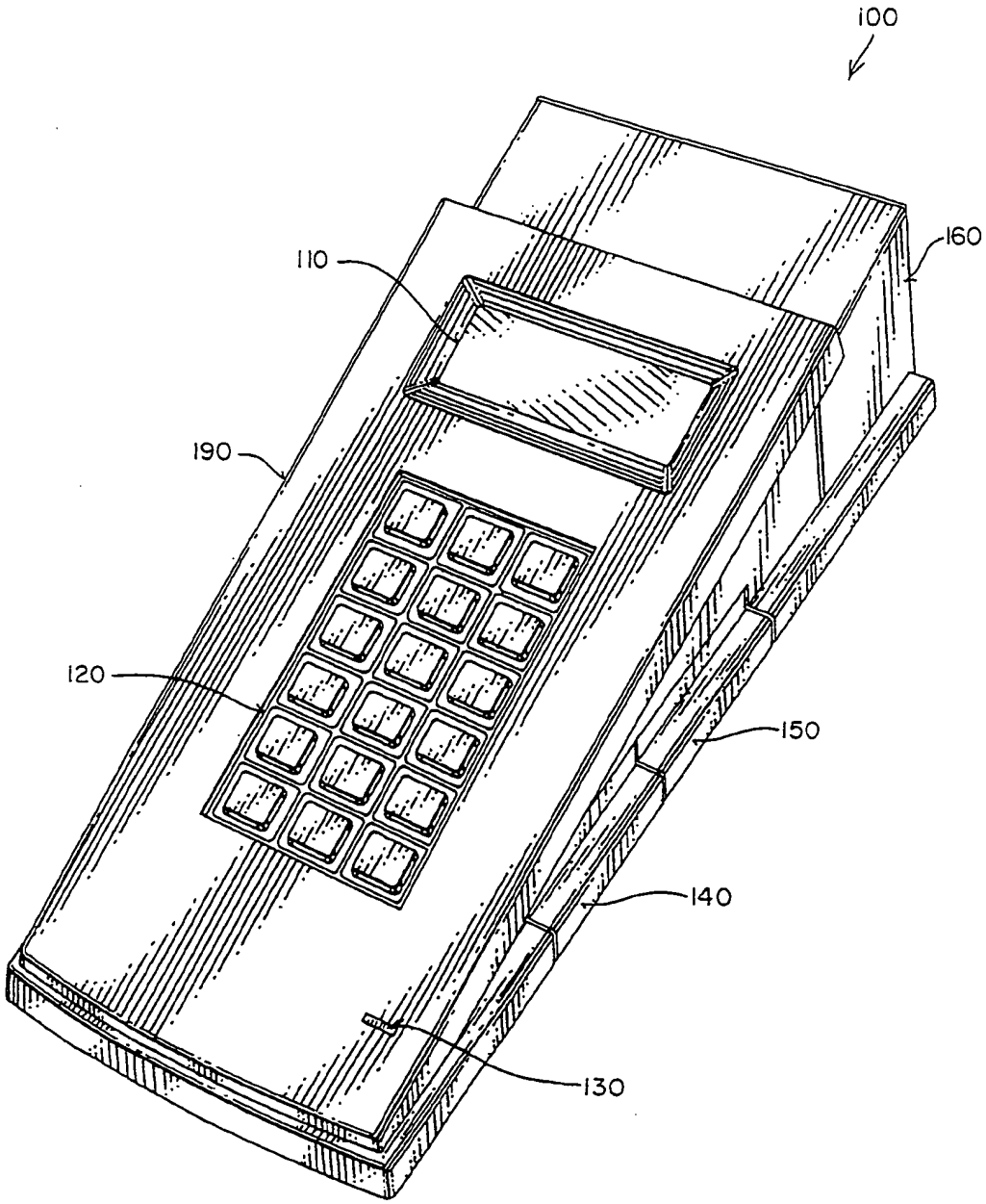
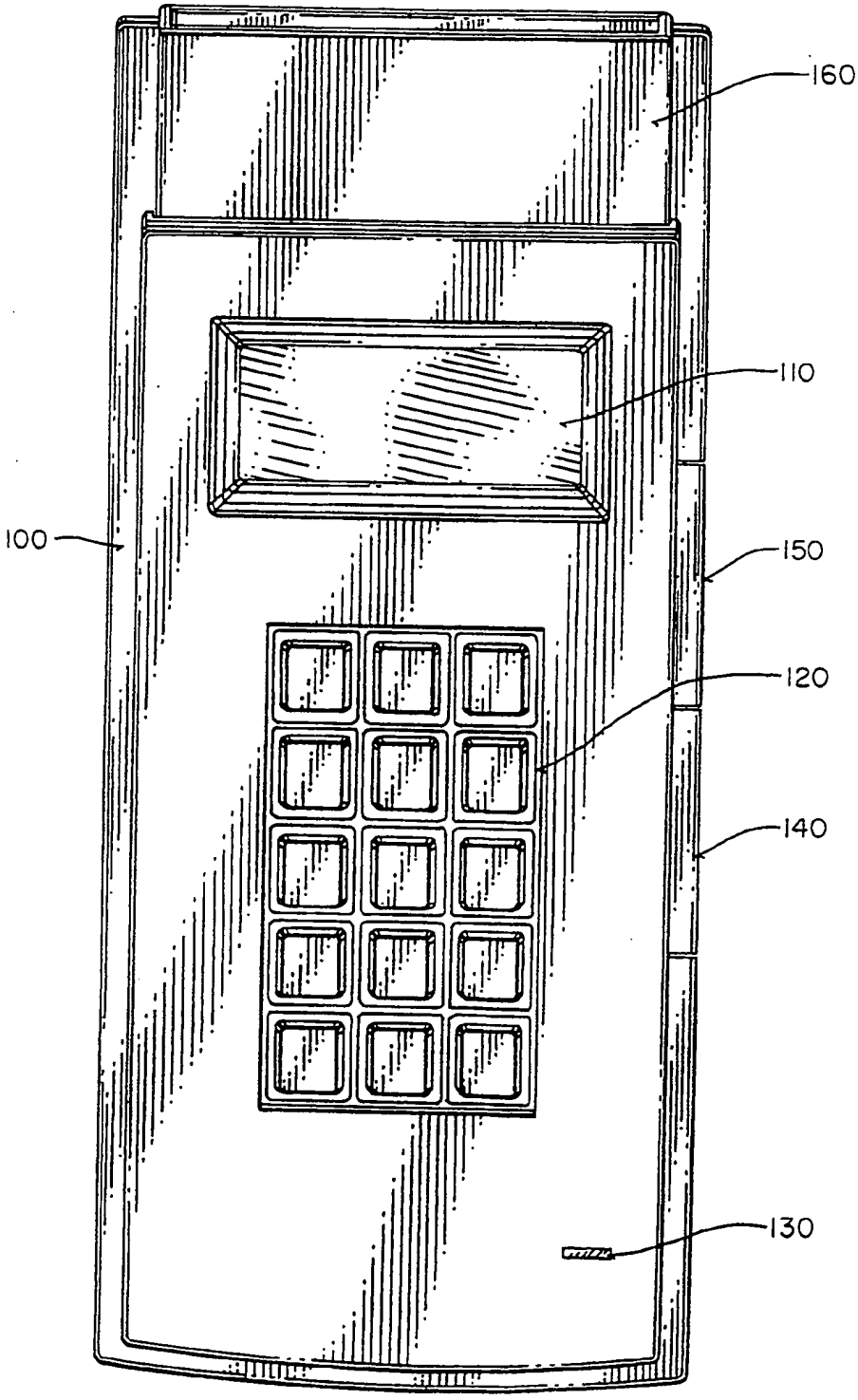


FIG 2



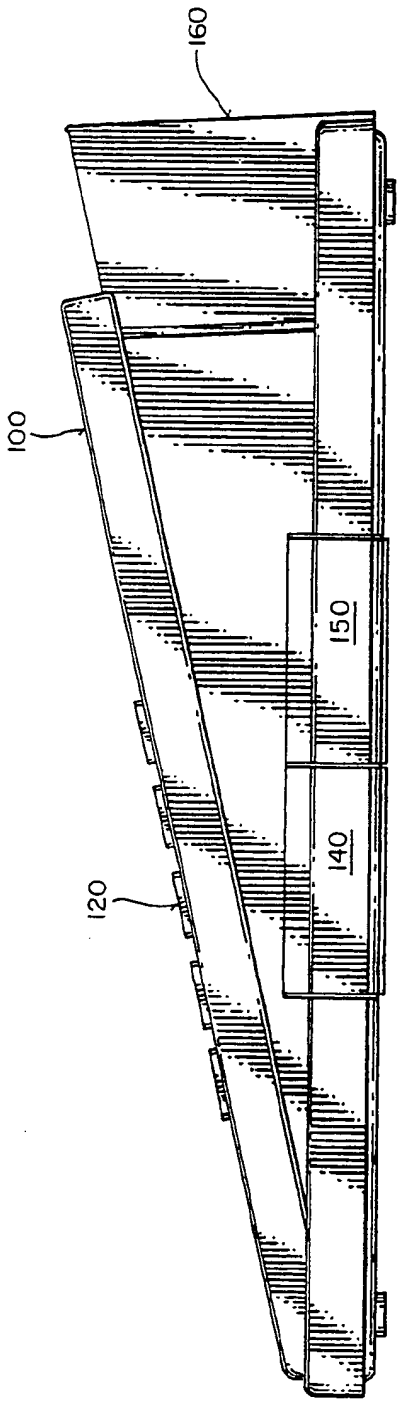


FIG. 3

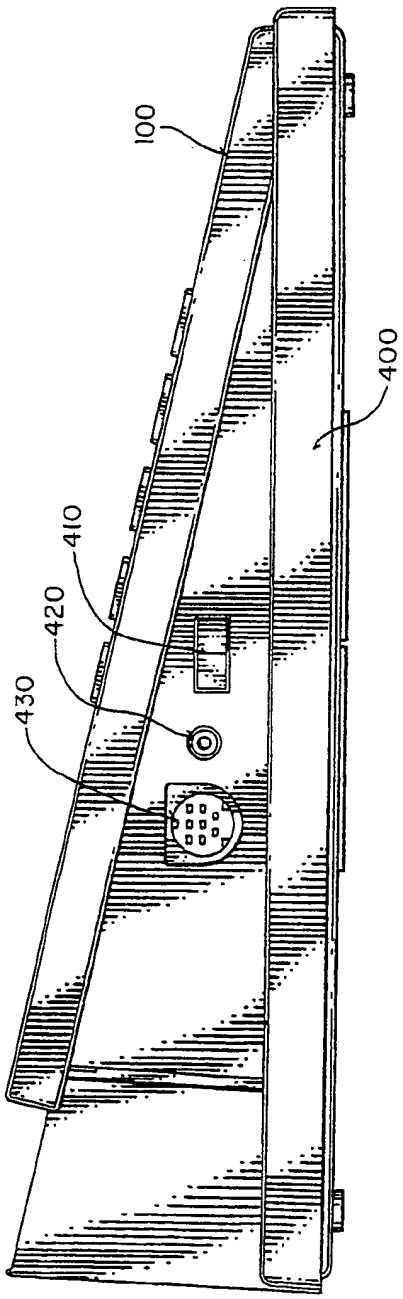


FIG. 4

FIG. 5

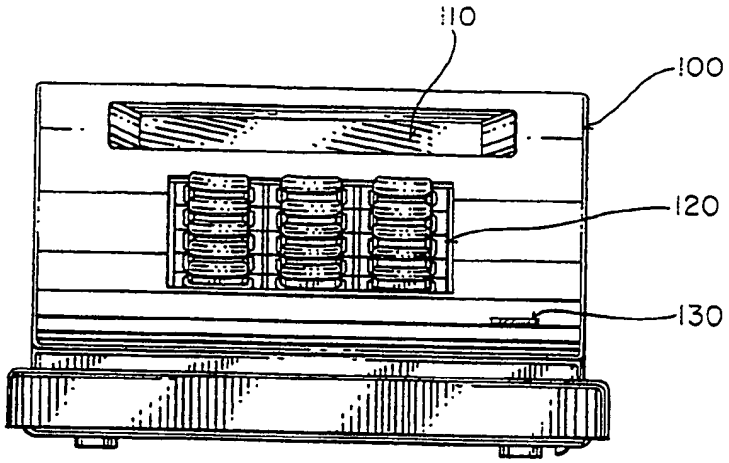
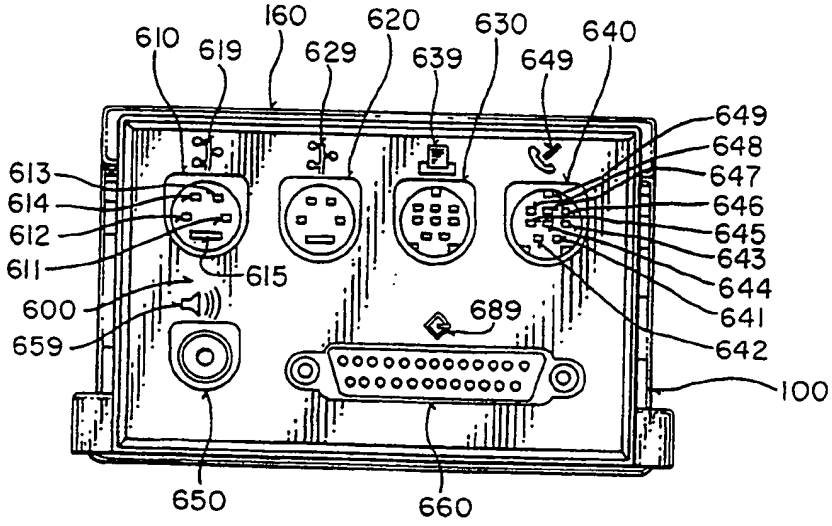

110
100
120
130

FIG. 6


610
619
160
629
620
639
630
649
640
649
648
647
646
645
643
644
641
642
613
614
612
611
615
600
659
689
100
650
660

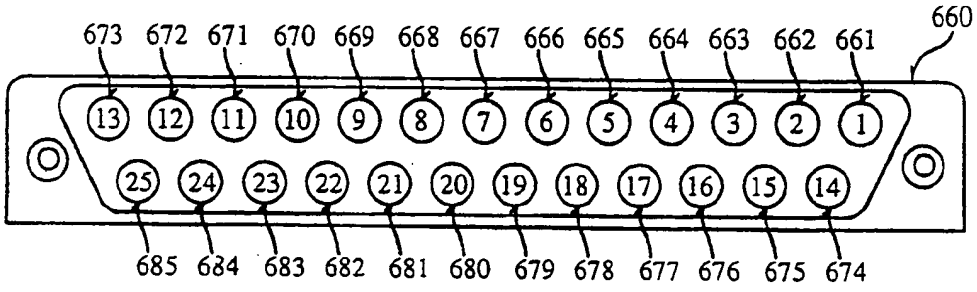

673 672 671 670 669 668 667 666 665 664 663 662 661
660
13 12 11 10 9 8 7 6 5 4 3 2 1
25 24 23 22 21 20 19 18 17 16 15 14
685 684 683 682 681 680 679 678 677 676 675 674

Figure 6a

FIG 7

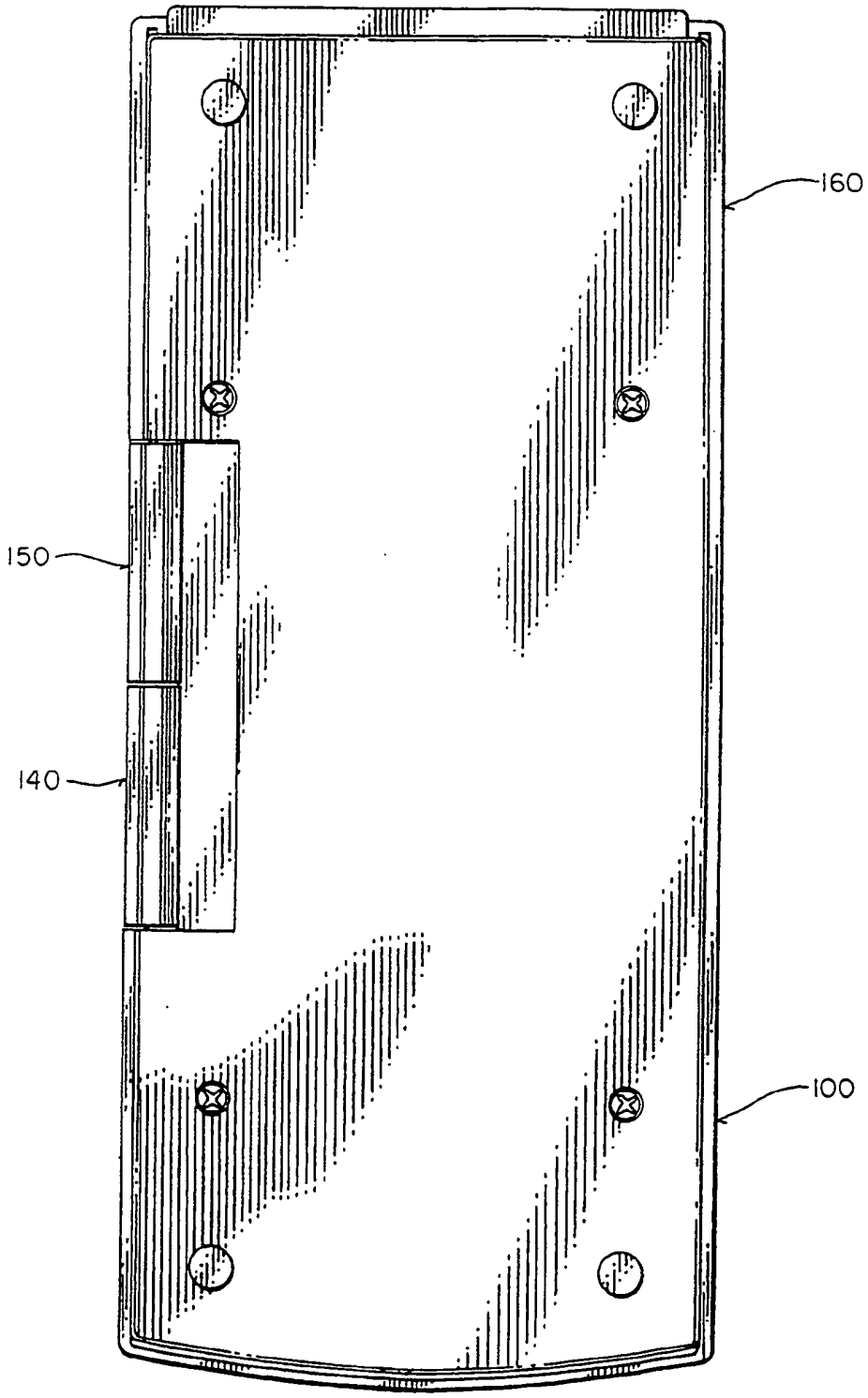
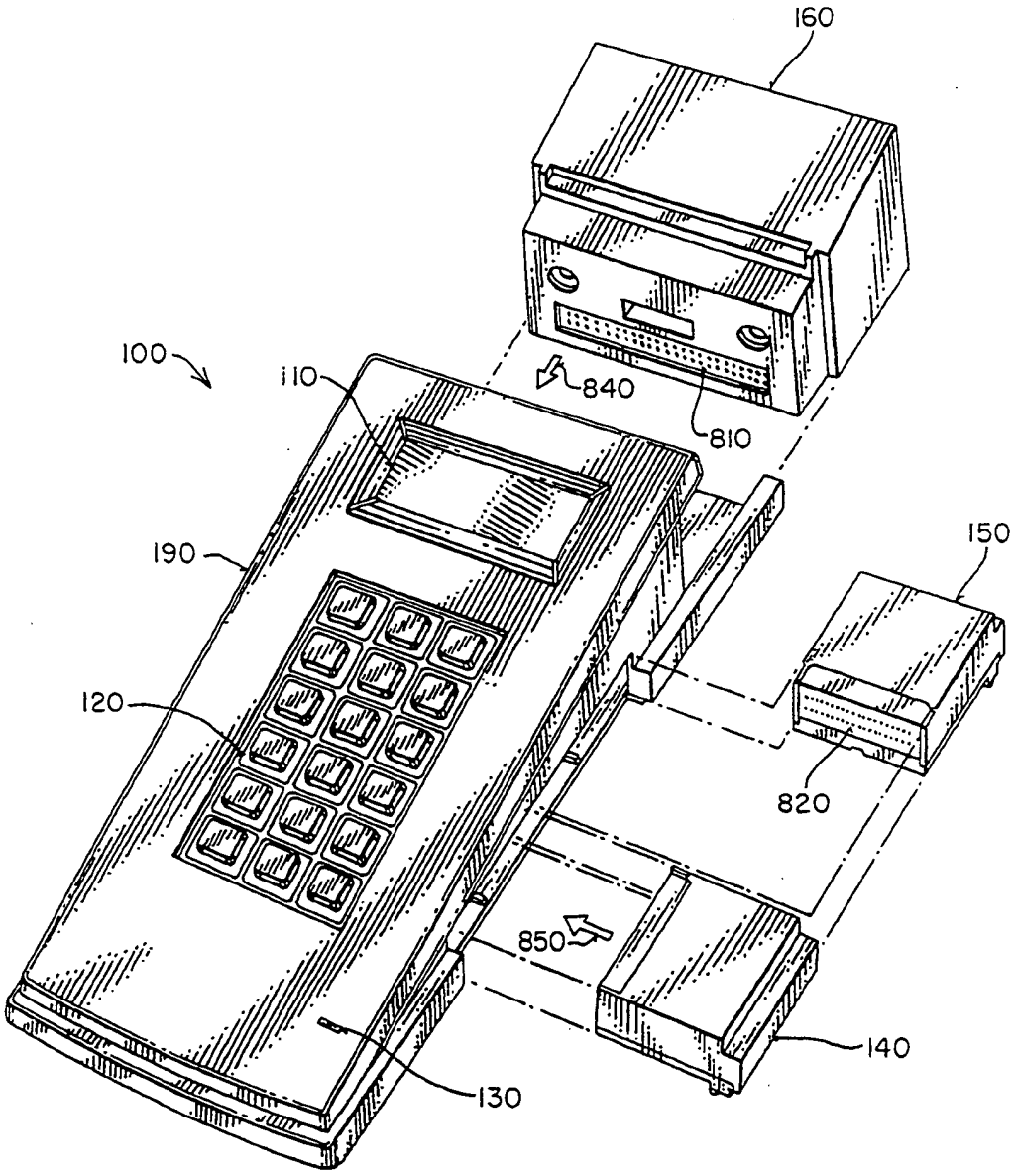


FIG 8



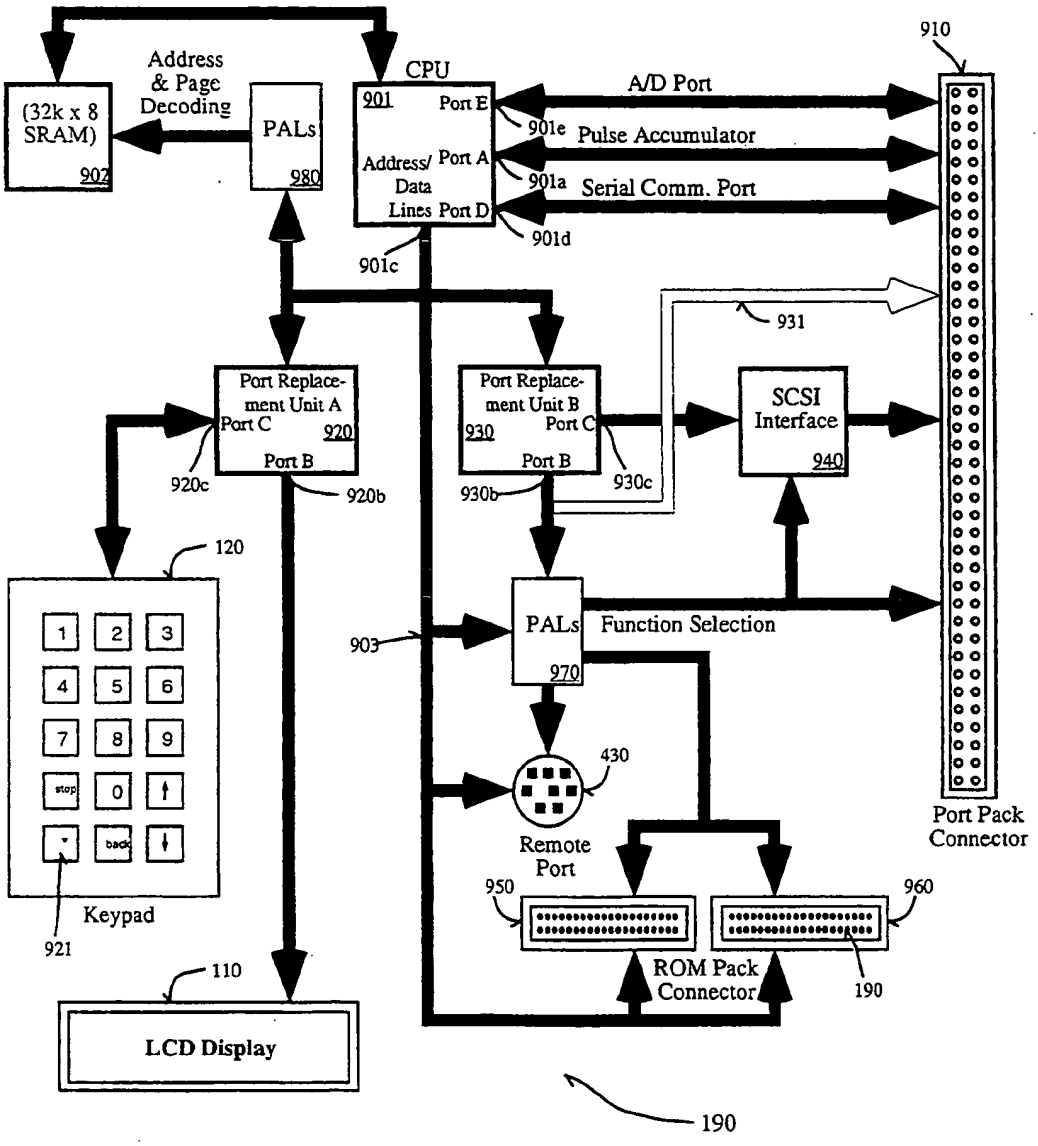


Figure 9 - Diagnostic Tool Base Unit

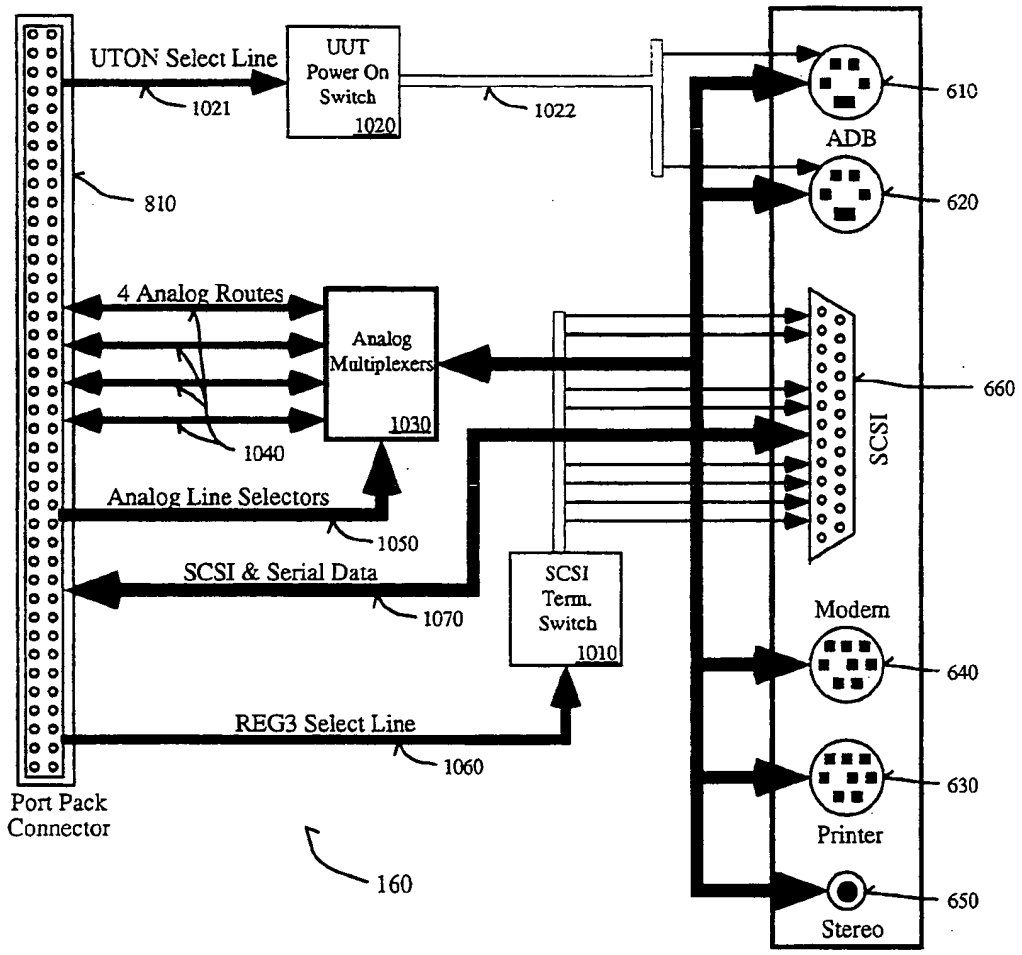


Figure 10 - Peripheral Port Pack

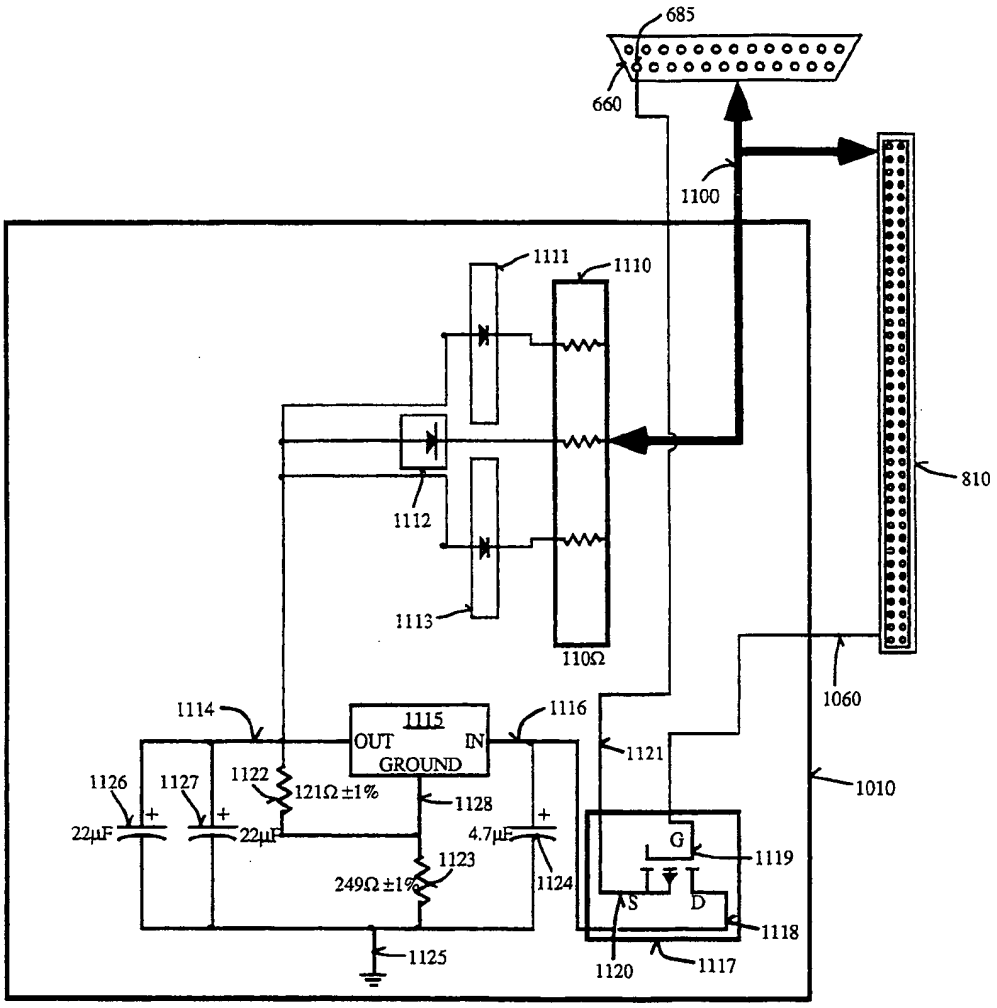

685
660
1100
1111
1110
810
1112
1113
110Ω
1060
1114
1115
OUT
IN
GROUND
1116
1121
1010
1126
1127
1122
121Ω ±1%
1128
22μF
22μF
4.7μF
1123
249Ω ±1%
1124
G
1119
S
D
1118
1125
1120
1117

Figure 11

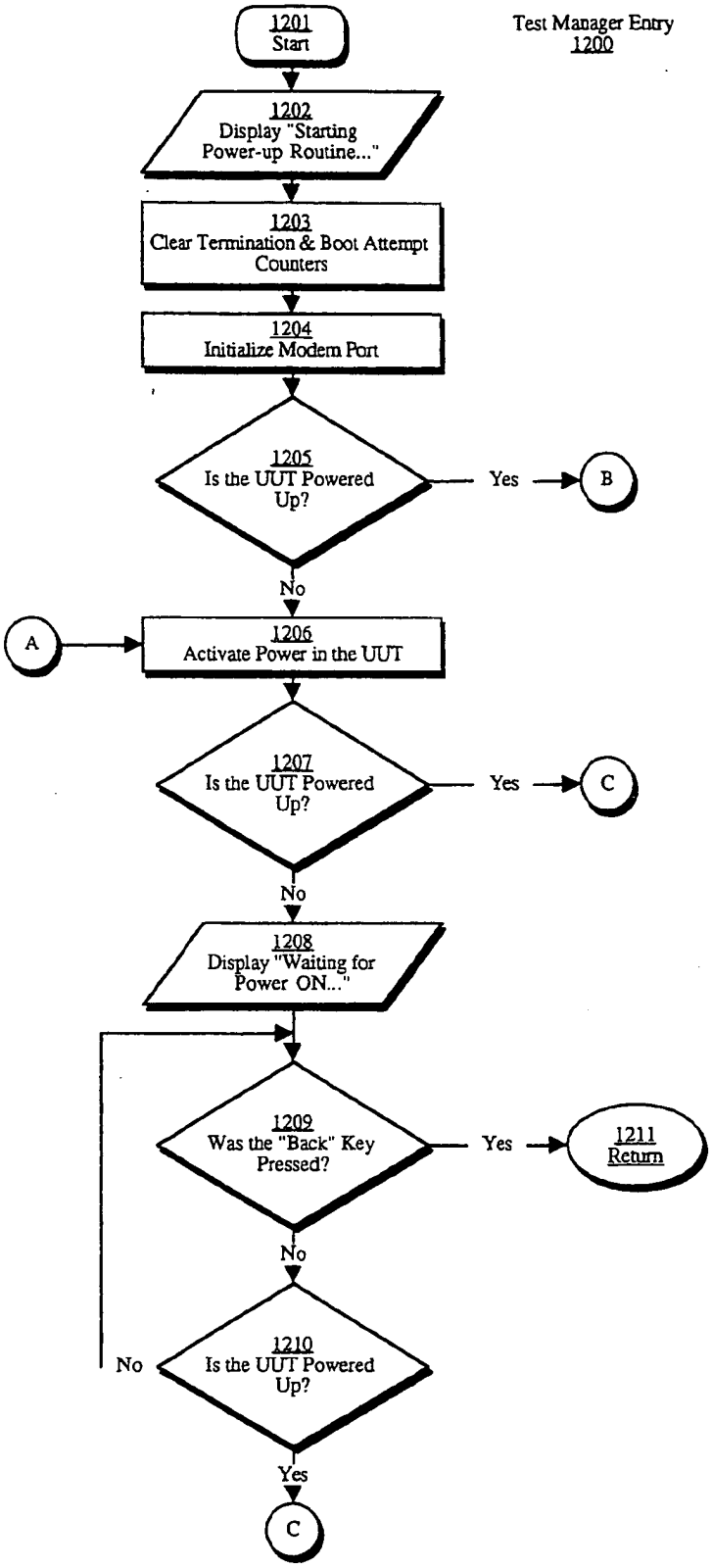

Test Manager Entry
1200
1201
Start
1202
Display "Starting Power-up Routine..."
1203
Clear Termination & Boot Attempt Counters
1204
Initialize Modem Port
1205
Is the UUT Powered Up?
Yes
B
No
A
1206
Activate Power in the UUT
1207
Is the UUT Powered Up?
Yes
C
No
1208
Display "Waiting for Power ON..."
1209
Was the "Back" Key Pressed?
Yes
1211
Return
No
1210
Is the UUT Powered Up?
No
Yes
C

Figure 12a

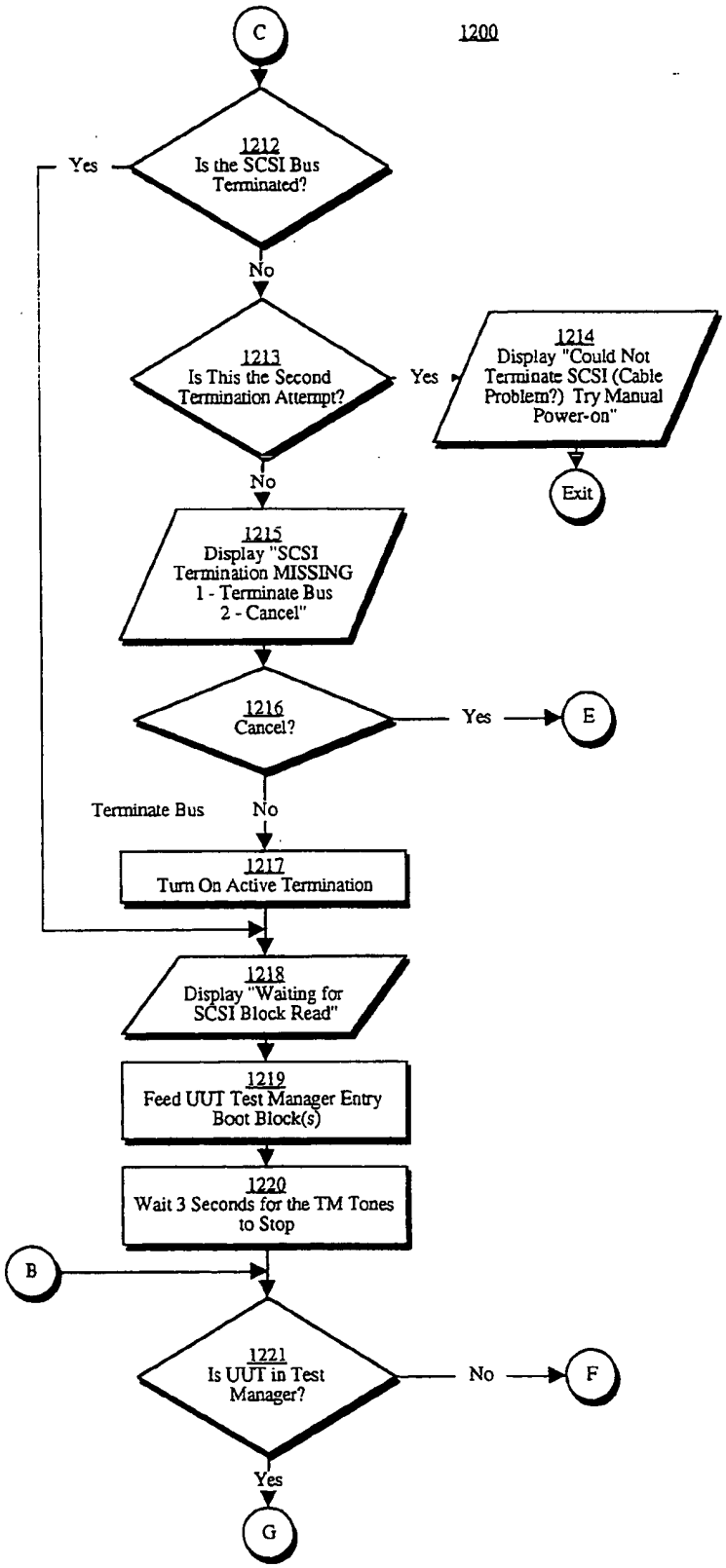

C
1200
1212
Is the SCSI Bus Terminated?
Yes
No
1213
Is This the Second Termination Attempt?
Yes
1214
Display "Could Not Terminate SCSI (Cable Problem?) Try Manual Power-on"
Exit
No
1215
Display "SCSI Termination MISSING
1 - Terminate Bus
2 - Cancel"
1216
Cancel?
Yes
E
Terminate Bus
No
1217
Turn On Active Termination
1218
Display "Waiting for SCSI Block Read"
1219
Feed UUT Test Manager Entry Boot Block(s)
1220
Wait 3 Seconds for the TM Tones to Stop
B
1221
Is UUT in Test Manager?
No
F
Yes
G

Figure 12b

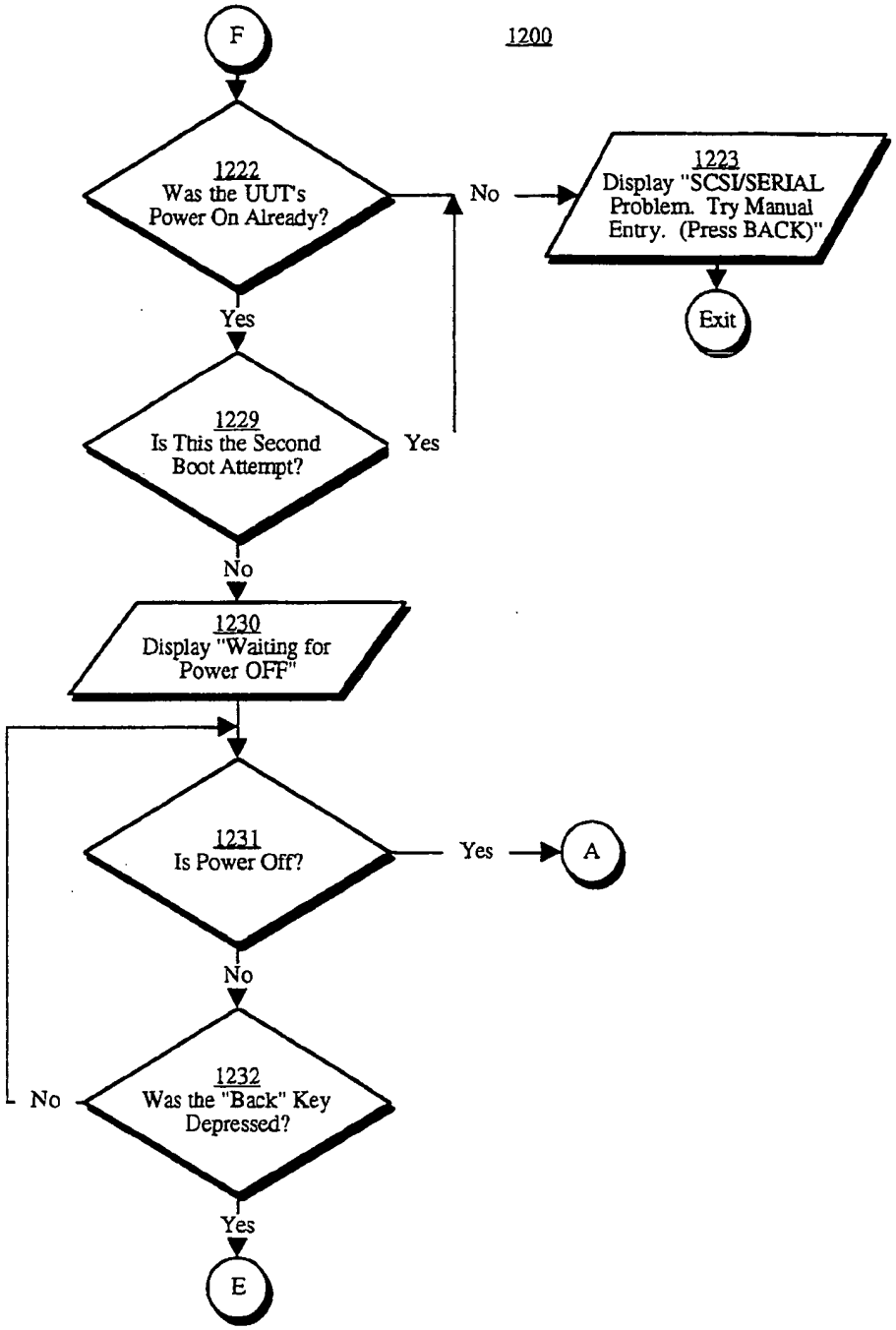

F
1200
1222
Was the UUT's Power On Already?
No
1223
Display "SCSI/SERIAL Problem. Try Manual Entry. (Press BACK)"
Exit
Yes
1229
Is This the Second Boot Attempt?
Yes
No
1230
Display "Waiting for Power OFF"
1231
Is Power Off?
Yes
A
No
1232
Was the "Back" Key Depressed?
No
Yes
E

Figure 12c

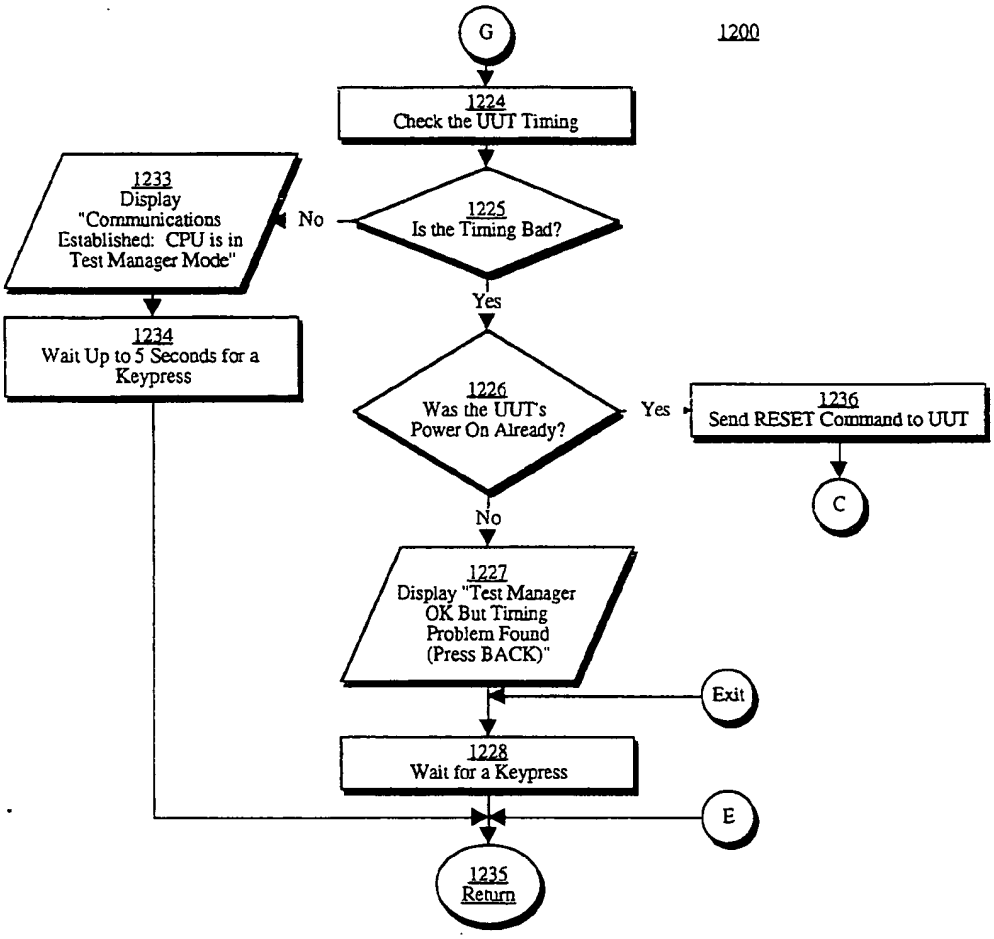
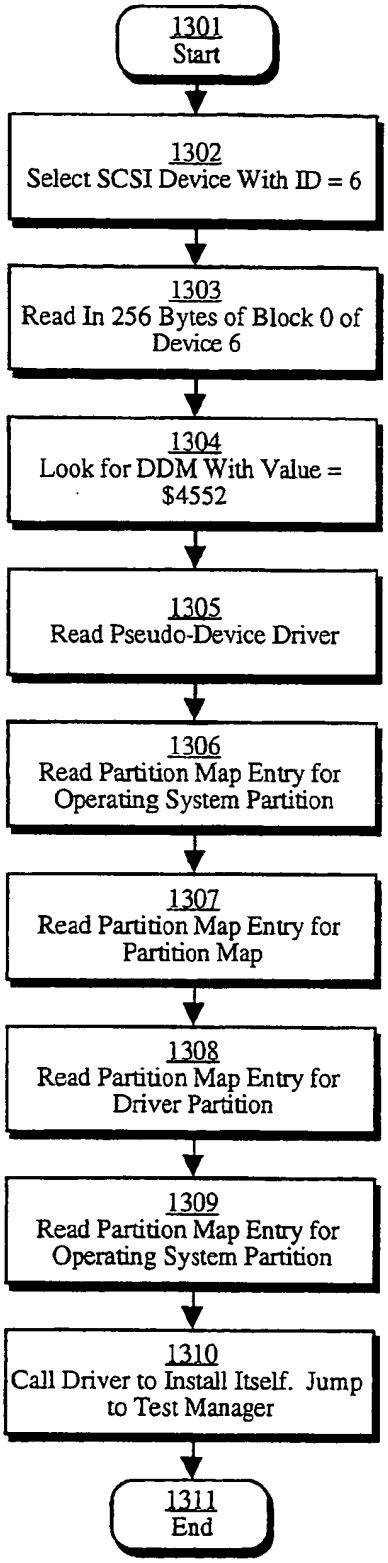

G
1200
1224
Check the UUT Timing
1225
Is the Timing Bad?
No
1233
Display
"Communications
Established: CPU is in
Test Manager Mode"
Yes
1234
Wait Up to 5 Seconds for a
Keypress
1226
Was the UUT's
Power On Already?
Yes
1236
Send RESET Command to UUT
C
No
1227
Display "Test Manager
OK But Timing
Problem Found
(Press BACK)"
Exit
1228
Wait for a Keypress
E
1235
Return

Figure 12d



SCSI TEST MANAGER STARTUP

1300

Figure 13

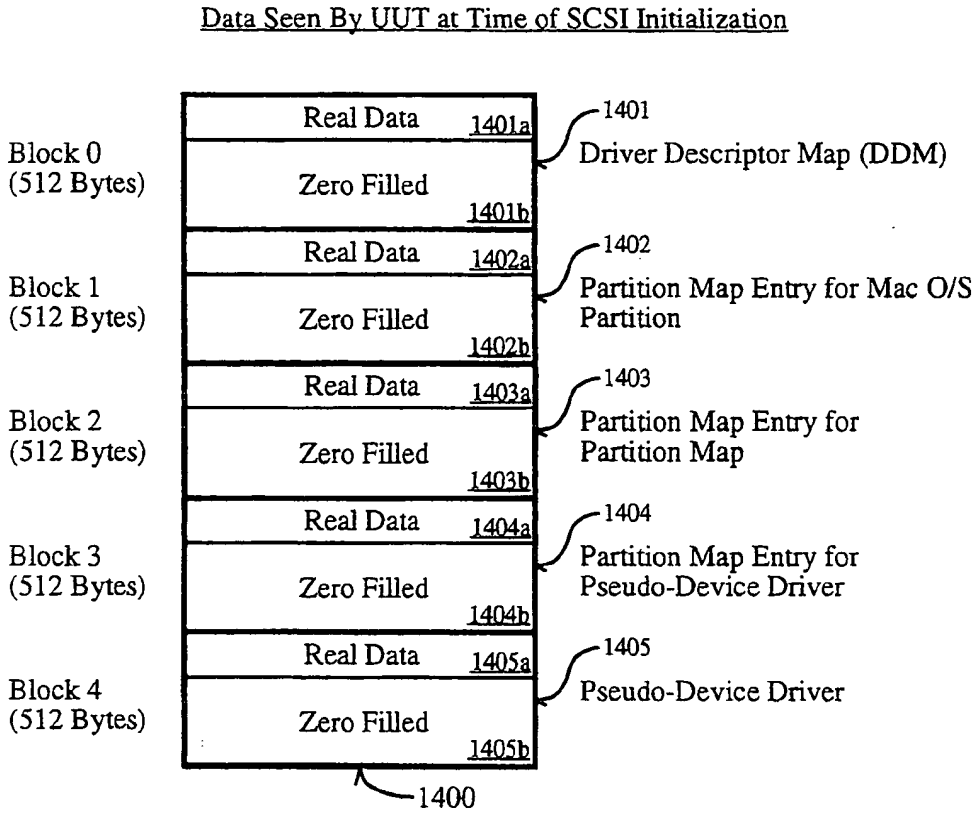

Data Seen By UUT at Time of SCSI Initialization
Block 0
(512 Bytes)
Block 1
(512 Bytes)
Block 2
(512 Bytes)
Block 3
(512 Bytes)
Block 4
(512 Bytes)
Real Data
1401a
Zero Filled
1401b
Real Data
1402a
Zero Filled
1402b
Real Data
1403a
Zero Filled
1403b
Real Data
1404a
Zero Filled
1404b
Real Data
1405a
Zero Filled
1405b
1400
1401
Driver Descriptor Map (DDM)
1402
Partition Map Entry for Mac O/S Partition
1403
Partition Map Entry for Partition Map
1404
Partition Map Entry for Pseudo-Device Driver
1405
Pseudo-Device Driver

Figure 14

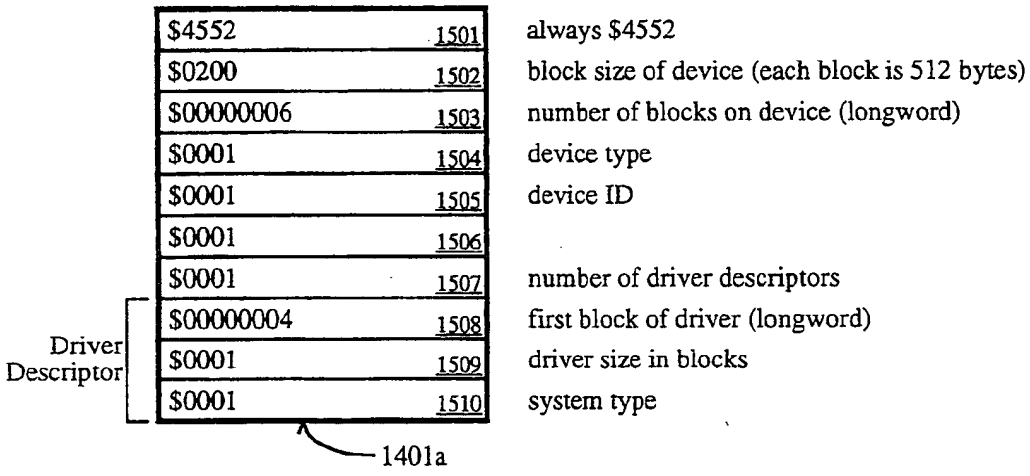


Figure 15 - Driver Descriptor Map

ACTIVE BUS TERMINATION DEVICE

This is a divisional of application Ser. No. 07/771,127, filed Oct. 3, 1991 now U.S. Pat. No. 5,357,519.

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention relates to the field of diagnostic methods and apparatuses for computer systems. More specifically, this invention relates to a device and method for diagnosing a faulty computer system without disassembling the system.

2. Description of Related Art

The diagnosis of faulty computer systems, at times, can be a difficult process. For systems which are incapable of powering up and performing a system bootstrap initialization process, determining which components are faulty requires substantial time and effort using a variety of methods. One method of diagnosing a faulty computer system is to remove components in the computer system one by one and replace them with components which are known to work. Such a "trial and error" approach not only requires an on-hand inventory of components which are functional, but also requires that the computer system be disassembled. This type of diagnosis requires substantial effort to disassemble and reassemble the system and does not necessarily identify the underlying defect which caused the failure.

Yet another method for diagnosing a faulty computer system is to couple various types of diagnostic apparatuses to individual components. This process requires some disassembly of the computer system to attach diagnostic probes to the individual components. Depending on the location of individual components, this process may take a lot of time to disassemble the faulty device. This also requires specialized tools to test individual components.

For a system which is able to power up and execute a system bootstrap initialization but does not function properly, diagnosis may be performed using a software-utility. Some types of utility programs may diagnose certain faulty components, but if the faulty components are required to operate the utility, the diagnosis of the system will be impossible. Even though the disassembly of the computer system may not be required to run such a diagnostic program, this program may be limited by the inability to test certain hardware components.

Some computer systems perform a software diagnostic self-test upon a system bootstrap initialization process. This type of routine tests various components in the system prior to operation to ensure that the system is operating properly. One such diagnostic routine is known as the "Test Manager" program which forms pan of the bootstrap initialization procedure of the Macintosh ® brand computer available from Apple Computer, Inc. of Cupertino, Calif. (Macintosh ® is a trademark of Apple Computer, Inc.). This initialization process includes, among other things, initializing versatile interface adaptor (VIA) circuits, serial communication controller (SCC) circuits, floppy disk drive integrated circuit controllers, small computer system interface controller (SCSI) circuits, and sound device circuits coupled to the system. These routines, which are embedded in read-only memory (ROM) contained within the system, require that the system be capable of activating system power and initializing. If the system cannot be initialized, then certain types of tests may not be able to be performed.

Another drawback of prior approaches to diagnosing computer systems is that different computer systems have a variety of interfaces and ports. Each port or interface requires a different connector and/or different circuitry to test these ports or interfaces. For instance, in addition to generic serial and parallel ports such as RS-232 standard ports or SCSI ports, some computer systems may have manufacturer-specific ports such as the Apple Desktop Bus (ADB) brand interface manufactured by Apple Computer, Inc. The wide variety of ports, interfaces and other coupling means for various systems makes it difficult to diagnose the many types of computer systems in the marketplace. Because the parameters of each interface and/or port must be known to the individual diagnosing the computer system, it is difficult for one person to diagnose a variety of machines. In addition, it is helpful if various types of connectors are available for coupling to the various systems for diagnosis. In summary, no single device possesses the necessary characteristics to diagnose various types of computer systems at all levels of operation.

SUMMARY AND OBJECTS OF THE INVENTION

One object of the present invention is to provide a device which can diagnose a computer system incapable of performing a system power up or bootstrap initialization process.

Another object of the present invention is to provide a device which performs nonintrusive diagnostics of a computer system.

Another object of the present invention is to provide a device which is capable of diagnosing various types of computer systems including system-specific connectors and interfaces.

Another object of the present invention is to provide a method and apparatus which facilitates diagnostics of a computer system with a minimal level of system-specific knowledge by a user.

These and other objects of the present invention are provided for by a diagnostic apparatus for testing devices such as computer systems, and computer system peripheral devices such as disk drives or printers. The apparatus comprises a main unit, the main unit having a central processing unit for executing instructions, issuing commands, and receiving data from a first device being tested. In a preferred embodiment, the apparatus comprises a display and keyboard for communicating with a user of the apparatus. The apparatus also has a first peripheral unit coupled to the main unit, the first peripheral unit having ports for interfacing with the first device, the first peripheral unit being interchangeable with a second peripheral unit for interfacing with a second device. The first peripheral unit may, for example, be used for testing one computer system such as a personal computer, and the second peripheral unit may be used for testing a second personal computer, disk drive, or printer. The apparatus also comprises a first non-volatile memory unit coupled to the main unit, the first non-volatile memory unit comprising a first set of tests for the first device. The first non-volatile memory unit is interchangeable with a second non-volatile memory unit comprising a second set of tests for a second device. These units are provided so that the user may test various types of hardware.

These and other objects of the present invention are provided for by a device for terminating a bus. In a preferred embodiment, the termination on a bus, for example a small computer system interface (SCSI) may be desired. The device comprises a first means for activating termination of the bus which, in a preferred embodiment, is a flag in a register. When set, the flag switches on SCSI termination, and when not set (e.g., cleared) there is no termination. The device further comprises a second means coupled to the first means for supplying power, the second means being activated upon activation of the first means. In a preferred embodiment, the second means is a p-channel MOS-FET. The device also has a third means coupled to the second means for limiting voltage received from the second means which is a low dropout voltage regulator. Lastly, the device has a fourth means coupled to the third means for limiting transient voltages on the bus, which prevents the bus from appearing as if any device is coupled to the bus.

These and other objects of the present invention are provided for by a means for remotely entering a diagnostic program in a computer system. This method comprises simulating a first device coupled to the computer system and simulating a driver for the first device coupled to the computer system. The computer system is caused to load the driver and then to call the driver to execute itself in the computer system. The driver contains a jump instruction to the diagnostic program causing the diagnostic program to then execute.

BRIEF DESCRIPTION OF DRAWINGS

The present invention is illustrated by way of example and not limitation in the figures of the accompanying in which like references indicate like elements and in which:

FIGS. **1** through **7** including **6***a*, show the external physical configuration of the diagnostic apparatus.

FIG. **8** shows a removable port pack and removable ROM packs in a disassembled configuration used in the diagnostic device of the preferred embodiment.

FIG. **9** is a block diagram of the base unit of the diagnostic device.

FIG. **10** is a block diagram of file port pack of the diagnostic device.

FIG. **11** is a schematic of the software controllable SCSI termination apparatus used by the diagnostic device.

FIG. **12***a*–**12***d* are flowcharts showing various ways to enter test routines used in the diagnostic device of the preferred embodiment.

FIGS. **13** through **15** show the SCSI device emulation used by the preferred embodiment to enter the Test Manager brand diagnostic program.

DETAILED DESCRIPTION

A device and method for diagnosis of computer systems is described. In the following description, for the purposes of explanation, numerous specific details are set forth such as circuitry, signal names, signal lines, and pan numbers are set forth in order to provide a thorough understanding of the invention. It will be obvious, however, to one skilled in the an that the invention may be practiced without these specific details. In other instances, well-known circuits, structures, and techniques have not been shown in detail in order to not unnecessarily obscure the present invention.

Physical Configuration of the Diagnostic Tool

FIG. **1** shows the external physical configuration of the preferred embodiment of the present invention. Diagnostic device **100** is generally used for testing the operation of computer systems, however, it may be used for testing individual components of computer systems such as hard disk drives, printers, monitors, or other peripherals. Diagnostic device **100** comprises several portions which are shown in their assembled and operating configuration in FIG. **1**. Diagnostic device **100** comprises a display **110** for presenting information to the user of the device. Display **110** is one of the many liquid crystal displays (LCD's) which are commercially available and well-known to those skilled in the art. Diagnostic device **100** further comprises a keypad **120** for indicating command selections and other information to device **100**. Further device **100** comprises a low-power light emitting diode (LED) **130** indicating when battery power in the unit is becoming depleted. In order to provide the utmost flexibility, device **100** comprises modules **140**, **150**, and **160** which are all removable from device **100** and interchangeable with other modules. **140** and **150** are removable "ROM packs" which provide different tests and different operating modes for the various computer systems or computer peripherals (hereinafter units under test or UUT's) which may be tested by device **100**. **160** is also a removable module and is known as a "port pack." Port pack **160** provides computer system ports for the various types of ports which may be present on a system being tested. ROM packs **140** and **150** and port pack **160** will be discussed in more detail below.

FIG. **4** shows one side **400** of device **100**. Side **400** of device **100** comprises a power switch **410** which is a single pole, single throw (SPST) rocker switch which is used to activate power to device **100**. Further, device **100** comprises an adaptor plug **420** which is a miniature dual conductor plug used for supplying 5 volt power to unit **100** via an integrated power supply adaptor cable. Diagnostic device **100** may also be powered by an internal battery when not coupled to a power supply via plug **420**. In addition, as shown on side **400** of FIG. **4**, device **100** comprises a female serial port **430** which is used for coupling device **100** to a modem (modulator/-demodulator) for communication over telephone lines. This provides the capability for device **100** to communicate with other devices **100** or UUT's which can be remotely tested. Connector **430** is a mini 8-pin serial port connector which conforms to EIA standard RS-232 signal conventions. The signals and pins on connector **430** are described in more detail in *Guide to the Macintosh Family Hardware*, Second Edition by Apple Computer, Inc. available from Addison-Wesley Publishing Company, Inc. (copyright 1990) (hereinafter "Hardware Guide") at pages 357–374. Various ports and detailed connections to port **430** will be discussed in more detail below.

Removable ROM Packs

For the greatest flexibility in testing various types of computer systems and other devices with diagnostic tool **100**, ROM packs **140** and **150** and port pack **160** are provided which are all removable from main unit **190** of device **100** as shown in FIG. **8**. These packs may be replaced with other packs which have different ports and/or different of tests. As is shown in FIG. **8**, port pack **160** may be coupled to main unit **190** via connector

810. Connector 810, in the preferred embodiment, is a Fujitsu FCN215Q080:G/O, 80-pin. male connector. A detailed description of the signals and the lines on connector 810 will be discussed below. 810 provides an interface with main unit 190 of device 100 so that communication with various peripheral ports on pack 160 may be accomplished. Port pack 160 may be coupled to main unit 190 by affixing 160 to 190 in the direction shown as 840 on FIG. 8. This mates connector 810 with a corresponding female connector on main unit 190 (not shown). In addition, 100 comprises removable ROM packs 140 and 150 which comprise non-volatile test routines and other device or system-specific diagnostic programs. As is shown in FIG. 8, each module such as 150 comprises a connector such as 820 which is mated to the main unit 190 to provide access to the test routines embedded in the non-volatile memory of each ROM pack 140 or 150. Each ROM pack such as 140 and 150 may be coupled to main unit 190 via a 40-pin Fujitsu FCN215Q040-G/O male connector which mates with a corresponding female connector on main unit 190 (not shown). A ROM pack such as 140 may be inserted into main unit 190 in a direction shown by arrow 850 in FIG. 8 to provide communication between main unit 190 and ROM packs 140 and 150.

Removable Port Pack

A more detailed view of the rear 600 of port pack 160 is shown in FIG. 6. 600 of 160 shows various ports contained in one port pack 160 which is designed for testing various models of the Macintosh brand computer system family manufactured by Apple Computer, Inc. of Cupertino, Calif. Port pack 160 is used for testing the models Macintosh SE, SE/30, and the Macintosh II brand family of computers including the IIx, IIcx, among others. It can be appreciated by one skilled in the an, however, that a port pack other than 160 may be plugged into base unit 190 of diagnostic device 100 in order to provide a different set of ports. In the preferred embodiment, port pack 160 has six ports. Base unit 190 may accommodate any number of ports depending on the configuration of the port pack. It will be appreciated by one skilled in the art, that any number of ports may be present on a peripheral port pack in various embodiments of the present invention. A detailed description of the ports available on port pack 160 of device 100 shown in FIG. 6 will now be discussed.

As is shown in FIG. 6, side 600 of port pack 160 comprises several ports which interface with Macintosh brand computer systems. Port pack 160 comprises two Apple Desktop Bus (ADB) brand connectors 610 and 620, two miniature 8-pin EIA RS-424 standard serial connectors 630 and 640, a two-conductor miniature audio connector 650 for coupling to audio ports, and a 25-pin female DB25 SCSI connector 650 for coupling to devices such as SCSI disk drives and disk drive ports on computer systems. ADB ports 610 and 620 are used for coupling with ADB brand ports on Macintosh computer systems for communicating various information to and from the computer via devices such as keyboards, trackballs, or mice. The pin assignments for 611 through 614 are shown in FIG. 6 and the corresponding signals are described with reference to Table 1 below.

TABLE 1

Signal Assignments for ADB Connector 610

Pin Number	Signal Name	Signal Description
611	ADB	Bidirectional data bus used for input and output. It is an open-collector signal pulled up to +5 V through a 470 Ω resistor on an Apple Macintosh brand computers main logic board.
612	POWER.ON	On the Macintosh II family, a key on the keyboard momentarily grounds this pin to pin 614 to switch on the power supply. On other Apple Macintosh brand computers this pin is not connected.
613	+5 V	+5 volts.
614	GND	Ground.

A more detailed discussion of the ADB signals and devices used in the preferred embodiment is discussed in *Hardware Guide* at pages 287–326. A more detailed discussion of signals issued by device 100 for performing diagnostics of a computer system is discussed below. The pin assignments for connector 620 are the same as those used on connector 610. 610 are 620 are indicated by their corresponding labels 619 and shown in FIG. 6, the ADB icons. Further, each of the connectors has a key 615 associated with it such as shown with reference to 610, to prevent improper insertion of cables into connectors 610 or 620.

Port pack 160 further comprises two serial ports 630 and 640. Serial ports 630 and 640 are used for coupling to a printer port or a modem port on a system as indicated by the appropriate labels 639 or 649. As discussed above, each of the serial ports in the preferred embodiment and on port pack 160 conform to the EIA RS-422 standard signal conventions which are described in more detail in *Hardware Guide* at pages 357–374. The pin assignments for serial port 640 is described with reference to Table 2 below. The signal assignments for the pins on 630 are the same.

TABLE 2

Signal Assignments for Mini 8-pin Serial Port Connector 640

Pin Number	Signal Name	Signal Description
641	HSKo	Handshake ouput. Driven inverted. Voh = 3.6 V; Vol = −3.6 V; Ri = 450 Ω
642	HSKi	Handshake input or external clock. Received uninverted. Vih = 0.2 V; Vil = −0.2 V; Ri = 12K Ω
643	T × D−	Transmit data (inverted). Driven inverted or tristated depending on the operation mode. Voh = 3.6 V; Vol = −3.6 V; Ri = 450 Ω
644	GND	Signal ground. Connected to logic and chassis ground.
645	R × D−	Receive data (inverted). Vih = 0.2 V; Vil = −0.2 V; Ri = 12K Ω
646	T × D+	Transmit data. Driven uninverted or tristated depending on the operation mode. Voh = 3.6 V; Vol = −3.6 V; Ri = 450 Ω
647	GPi	General-purpose input. Vih = 0.2 V; Vil = −0.2 V; Ri = 12K Ω
648	R × D+	Receive data. Received uninverted. Vih = 0.2 V; Vil = −0.2 V; Ri = 12K Ω

The two remaining ports on port pack 160 of the preferred embodiment are shown as 650 and 660 in FIG. 6. 650 is a standard 2-conductor female monaural audio

miniature jack used for diagnosing the sound capabilities of the device being tested. This is indicated by the sound label icon **659** above the sound jack. The processing of the signals received from sound jack **650** is discussed below.

Lastly, **160** comprises SCSI connector **660** which is a standard 25-pin female DB25 SCSI connector which is discussed in more detail in *Hardware Guide* at pages 375–396. SCSI connector **660** is labelled by icon **669** as shown in FIG. **6**. The circuitry coupled in main unit **190** to port pack **160** and thus to connector **660** is the discussed in more detail below. Pin outs on connector **660** are assigned as shown in FIG. **6a** with reference to Table 3 below.

TABLE 3

Signal Assignments for SCSI Connector 660

Pin Number	Signal Name	Signal Description
661	/REQ	Request for a REQ/ACK data transfer handshake
662	/MSG	Indicates the message phase
663	/I/O	Controls the direction of data movement
664	/RST	SCSI data bus reset
665	/ACK	Acknowledge for a REQ/ACK data transfer handshake
666	/BSY	Indicates whether SCSI data bus is busy
667	GND	Ground
668	/DB0	Bit 0 of SCSI data bus
669	GND	Ground
670	/DB3	Bit 3 of SCSI data bus
671	/DB5	Bit 5 of SCSI data bus
672	/DB6	Bit 6 of SCSI data bus
673	/DB7	Bit 7 of SCSI data bus
674	GND	Ground
675	/C/D	Indicates whether control or data is on the SCSI bus
676	GND	Ground
677	/ATN	Indicates an attention condition
679	/SEL	Selects a target or an initiator
680	/DBP	Parity bit for SCSI data bus
681	/DB1	Bit 1 of SCSI data bus
682	/DB2	Bit 2 of SCSI data bus
683	/DB4	Bit 4 of SCSI data bus
684	GND	Ground
685	TPWR	+5 volts terminator power

Circuitry of the Diagnostic Tool

A block diagram of the main unit of diagnostic apparatus **190** is shown with reference to FIG. **9**. Central processing unit **901** of diagnostic tool **100** is a 68HC11E9 microcontroller manufactured by Motorola, Inc. of Schaumburg, Ill. The HC11E9 version of the microcontroller comprises the following features which are useful for implementing certain features of the preferred embodiment:

- 12 kilobytes of internal programmable read-only memory (PROM);
- 512 bytes of internal read-only memory (RAM);
- 512 bytes of internal electrically erasable programmable read-only memory (EEPROM);
- a serial communication interface with 32 selectable baud rates;
- a serial peripheral interface;
- an 8-channel, 8-bit analog to digital (A/D) converter; and
- an 8-bit pulse accumulator/generator system.

A detailed description of the M68HC11E9 CPU 901 used in the preferred embodiment may be found in the publication M68HC11 Reference Manual, Revision 1, by Motorola, Inc. of Schaumburg, Ill. (1990). CPU **901** is docked by a 9.8304 MHz crystal which internally divides to a 2.45676 MHz "machine cycle" timing reference for CPU **901** and external devices. CPU **901** is set into a "special bootstap" mode for updating and testing its internal EEPROM. During the normal operation of the preferred embodiment, CPU **901** is run in its expanded multiplex mode wherein dedicated ports B and C on CPU **901** are converted to 16-bit multiplex address and 8-bit data lines ADO through A/D7 and A8 through A15.

In addition to the onboard non-volatile memory and RAM available to CPU **901**, diagnostic base unit **190** further comprises a 32-kilobit by 8-bit static random access memory (SRAM) **902** which is coupled to CPU **901**. Memory **902**, in a preferred embodiment is a 32-kilobit by 8-bit SRAM, pan No. 60L256 manufactured by Motorola, Inc. of Schaumburg, Ill. SRAM **902** is for storing additional information needed by CPU **901**, such as a buffer area for various ports and for use as a scratch pad memory. As is shown in FIG. **9**, pulse accumulator circuitry of CPU **901** is coupled to port pack connector **910** via port A **901***a* to provide communication with various circuitry contained in the peripheral port pack **160** as shown in FIG. **8** via connector **810**. Port A **901***a* doubles as the pulse accumulator/timer circuit and I/O lines for CPU **901**. These lines provide the input capture and output capture compare functions for time signal measurements and generation to and from base unit **190**. Port A **901***a* is used for handling the UUT's input/output (I/O) signals. PAL's **980** provide the necessary address decoding to access SRAM **902**.

Further, port E **901***e* of CPU **901** is coupled to connector **910** which is used as an analog to digital (A/D) port. Port E **901***e* is used for measuring various voltages from the UUT including the battery and the power supply on the UUT. These are measured via channels **1**, **2**, **3**, **4**, and **5** of the A/D converter internal to CPU **901** coupled to port E **901***e*. Certain input signals are fed through voltage divider circuits and then through high impedance op amps. The divider and op amp circuit are used to measure voltages up to twice the +5 volt reference provided at V_{RH} such as for the sound connections and certain serial port lines. This increases the gain of certain input signals prior to digitizing by CPU **901**. The low voltage reference V_{LH} for the A/D converter is tied to ground.

Address/data lines **901***c* of CPU **901** are coupled to two port replacement units **920** and **930** which, in the CPU **901**'s expanded mode, allow additional ports to be made available to CPU **901**. When operating CPU **901** in the expanded mode ports B and C of the 68HC11 processor are reconfigured to operate as multiplexed address/data lines A/DO through A/D7 and A8 through A15. Of course, it will be appreciated by one skilled in the art, that other microcontrollers which do not operate in this manner may not need port replacement units such as **920** and **930**. Port replacement unit A **920** is used for controlling keypad **120** and LCD display **110** of diagnostic base unit **190**. Address/data lines of port **901***c* are further coupled to a second pot replacement unit **930**. **930** is coupled to a SCSI interface **940** which is a standard 53C80 SCSI interface integrated circuit manufactured by NCR Corporation for providing SCSI transfers to and from port pack **160** as shown in FIG. **1**.

Address/data lines **903** of CPU **901** are also coupled to ROM pack connectors **950** and **960** which are standard 40-pin CN215J040-G/O female connectors manufactured by Fujitsu Corporation. These provide the

coupling between the ROM packs **140** and **150** and CPU **901** for hardware specific diagnostics. Communication with ROM packs **140** and **150** is provided via port replacement unit B **930** and PAL's **970**. PAL's **970** are also coupled to remote port **430** for serial communications to perform remote diagnosis of peripherals or computers over telephone lines via modem.

When CPU **901** is operated in its multiplexed or expanded mode, general purpose I/O ports B and C of CPU **901** configured to operate as multiplexed address-/data lines. Port replacement unit A **920** is coupled to the address/data multiplex lines of port **901***c* on CPU **901** in order to provide communication between keypad **120**, LCD display **110** and unit **190**. Address lines A8 through A11 are processed through a PAL to provide the necessary decoding to generate a chip select signal (/CS) in order to indicate which of the two port replacement units is being accessed. Port replacement unit **920** enables communication between keypad **120**, LCD display **110**, and CPU **901**. Port B **920***b* of port replacement unit **920** is used for communication with LCD display **110**, except for 3 bits of port B which are used for controlling various functions in port pack **160**. The 4 most significant bits of the control information passed through port B are used for controlling display **110**. In the preferred embodiment, display **110** is an LM2433A4C16B dot matrix 4 line by 16 character liquid crystal display module available from Densitron International Corporation. Display **110** requires eight bits of information to generate a character. 4 bits (a nibble) of each datum contained in the control register of port B is written individually to the memory mapped region for LCD display **110** at a time. The high order nibble data is first sent for the display, followed by the low order nibble data in sequential order. When LCD display **110** receives the high nibble of an instruction or display character, it latches the data until it receives the low order nibble data and then the data (character or instruction) becomes available for display.

The remaining four bits of the port B control register **920***b* of port replacement unit A **920** are used for various control functions. Bit **0** (the least significant bit) of port B **920***b* is used to control the command/display mode for LCD display **110**. Bit **1** is used to control the read and write direction of LCD display **110**. The two remaining bits of port B **920***b* are used for controlling the UUT. Bit **2** is used to initiate a "power on" sequence in the Macintosh II brand family of computers available from Apple Computer, Inc. of Cupertino, Calif. via port pack unit **160** as discussed with reference to Table 1 above. Bit **3** of port B **920***b* of port replacement unit A **920** is used to supply power to port pack connector **910** during data transfers as a means of prolonging the battery life of diagnostic tool **100**.

Port C **920***c* of port replacement unit A **920** is used for receiving information from keypad **120**. Keypad **120**, in one embodiment, is a 15-key 9-pin custom silicon rubber keypad. **120** may be one of any number of keypads which are commercially available. The five most significant bits of the keypad memory map register are used for receiving data from keypad **120** in a manner well-known by those skilled in the an. The three least significant bits of the control register are used for enabling-/disabling the columns on the keypad for receiving data input from keypad **120**. Key **921** on keypad **120** (the "*" or "command" key) is used for generating an interrupt request to CPU **901** that a coined is being issued through keypad **120**. This is coupled to the maskable interrupt line of CPU **901** to indicate the issuance of a command.

A second port replacement unit B **930** shown in FIG. **9** is used for providing communication with ROM packs **140** and **150**, and port pack **160** in the preferred embodiment. When port replacement B **930** is selected via the chip select signal, it is used for communicating with SCSI interface **940**, and ROM packs **140** and **150**. It is also used for providing control to a UUT serial port **901***d* and for communication over remote port **430**. It can be appreciated by one skilled in the art, however, that any number of computer systems and/or ports may be used depending on the configuration of the port pack, such as **160**, which is coupled to diagnostic base unit **190**. Port C **930***c* of port replacement unit B **930** provides communication with SCSI controller **940**, which is, in turn, coupled to port pack **160** via connector **910**. This provides communication with SCSI devices coupled to a port such as **660** shown in FIG. **6**. Port C **930***c* of port replacement unit B **930** acts as a bidirectional interface to controller chip **940**, but also provides a serial communications port selector line for communication with remote port **430**. In addition, a sleep select pin (/TIRED) is coupled to port C which is used for putting diagnostic apparatus **100** into a mode in which the minimum power is consumed (known as a "sleep" mode). Power consumption is minimized by deactivating the display, and stopping execution of all functions except monitoring keyboard interrupts in a manner well-known to those skilled in the art. SCSI data is memory mapped into RAM area **902** providing bidirectional data transfers between port replacement unit B **930** and SCSI communications chip **940** for reading and writing data between UUT's. Signals for controlling SCSI transfers between communications chip **940** and port replacement unit **930** are well-known to those skilled in the art and are used for SCSI transfers via **940** in the preferred embodiment. One function provided by the preferred embodiment is the use of a software-controllable SCSI termination of the UUT for simulating the presence of device(s) coupled to the UUT. This termination is activated by setting a bit in a control register of port B. Diagnostic base unit **190** can also simulate a SCSI device to a UUT by simulating the presence of partitions, and file allocation tables for performing other types of testing.

Port B **930***b* of port replacement unit B **930** is used as general purpose port for external ROM and RAM page control, as well as ROM pack and serial EEPROM selectors. Address lines A12 through A15 on port **901***c* of CPU **901** are remapped using PAL's **970** to provide the appropriate memory mapping in CPU **901**'s random access memory. Each individual ROM pack is selected via a select bit contained within the control register for the ROM packs. This selects which connector **950** or **960** is accessed for data, to access tests to residing in ROM pack **140** or ROM pack **150**. Each ROM pack such as **140** or **150** may individually contain hardware specific tests.

Diagnostic base unit **190** lastly comprises a serial communication port which is coupled to port D **901***d* of CPU **901**. Three separate serial communication lines are provided through port **901***d* which are activated by two lines on port B **930***b* of port replacement unit **930**. When both of the control bits are cleared, the serial communications port **901***d* is put into a loop back test mode. Either one of these signals being activated enables one of two sets of serial communication transmit and re-

ceive lines. Both bits being set enables a third set of transmit and receive lines. Port D **901***d* of CPU **901** is coupled through an RS-232 driver receiver chip (Motorola part No. MC145407—not shown) which is then coupled to port pack connector **910**. The third set of serial transmit and receive lines are used for communication over remote port **430** for remote UUT testing. The first set of transmit and receive lines are the primary communication means by which the preferred embodiment communicates with software diagnostics embedded in certain types of computer systems, such as the Macintosh brand family of computers. These embedded software diagnostics are collectively known as the "Test Manager." The second set of transmit and receive lines are used for the loop back communication port signals for serial printer ports used in the Macintosh SE and Macintosh II brand family of personal computers.

Standard RS-232 signals are used by diagnostic device **190** for communication with UUT's. However, computer systems such as the Macintosh family of personal computers are capable of using EIA RS-422 hardware protocols which are currently unused by the communications ports directly. The unused lines of the RS-422 lines are selectively sampled using the A/D multiplexers on port pack **160** which, as discussed in more detail below, are connected to port E **901***e* on CPU **901** via port pack connector **910**. A detailed description of one peripheral port pack **160** which may be coupled to diagnostic base unit **190** in one embodiment will now be discussed in more detail with reference to FIG. **10**.

Circuitry of the Port Pack

A block diagram of peripheral port pack **160** which may be used in one embodiment of the present invention is shown and discussed with reference to FIG. **10**. As was discussed with reference to FIG. **6** above, port pack **160** comprises a series of ports **610** through **660** which are coupled via connector **810** to diagnostic base unit **190** in order to test specific types of UUT's. Peripheral port pack **160** has an FCN215Q080-G/O 80-pin male connector **810** manufactured by Fujitsu, Inc. which mates with connector **910** on base unit **190**. A control register of CPU **901** is used to select, via analog multiplexer **1030** shown in FIG. **10**, analog signal channels for measurement. Analog multiplexers **1030** ire controlled via analog line selectors **1050** which are coupled to port A **901***a* of CPU **901**. Another signal line coupled to port A **901***a* on CPU **901** is used for enabling SCSI termination. This is coupled via line **1060** from connector **810** to termination circuitry **1010**.

In the preferred embodiment, analog multiplexers **1030** comprise four 74HC4051's available from Motorola, Inc. The selection of each analog MUX **1030** enables any or all of lines **1040** for conversion and measurement of voltage values by base unit **190**. Also, SCSI and serial data lines **1070** from port pack **160** are coupled to connector **810** for communication with the serial and SCSI circuitry within base unit **190**. An additional feature provided by port pack **160** is UTON select line **1021**. When activated, this line activates power to certain types of UUT's. UTON select line **1021** is coupled to circuitry **1020** to momentarily ground pins **612** and **614** on connectors **610** and **620** in order to initiate a remote power up of the UUT as discussed with reference to Table 1 above. For other types of UUT's not having such a desion, the user must activate power manually.

For sampling various voltage levels on connectors **610** and **620** coupled to the UUT, pins **611** and **614** are coupled through analog multiplexers **1030** to connector **810**. In addition, for communicating with the UUT, connectors **610** and **620** are coupled to lines **1070** and then to connector **810**. Communication is thereby provided between the UUT via connectors **610** and **620** and diagnostic base unit **190** for performing various tasks. Communication via ports **610** and **620** is described in more detail with reference to *Hardware Guide* at pages 287–326.

Serial and SCSI communication is performed with the UUT through connectors **630**, **640**, and **660** is accomplished via the remaining lines **1070** coupled to connector **810**. Sampling of voltages on these connectors is also provided in a similar manner as discussed with regard to connectors **610** and **620** via analog MUX's **1030** across lines **1040**. Sound capabilities of the UUT, in certain systems, may also be tested in this manner. Other analog voltages, as may be appreciated by one skilled in the an, may be sampled in similar manners.

Software Contollable SCSI Bus Termination

A detailed description of the software controllable SCSI termination circuitry **1010** of the preferred embodiment will now be discussed with reference to FIG. **11**. The lines coupled from connector **810** to SCSI connector **660** are routed through **1010** for providing the bus termination. Five volts is received on line **1121** from pin **685** on connector **660** to circuitry **1010**. One signal line, REG3 **1060** from base unit **190**, is provided which activates termination. The remaining SCSI lines **1100** from connector **660** are coupled to the 110 ohm resistors of resistor packs **1110**, and resistor pack **1110** is coupled to the cathode on diodes in either of diode packs **1111**, **1112**, or **1113**. The anodes of all the diodes in diode packs **1111**, **1112** and **1113** are tied together and connected to an output line **1114** of the low drop out positive adjustable voltage regulator **1115**. Voltage from **1117** is fixed by resistors **1122** (121Ω) and **1123** (249Ω) which are coupled to ground on **1128** on **1115** and the output **1114** through a series of capacitors (**1126** and **1172** and **1124**) tied to ground **1125**. The input **1116** of device **1115** is coupled to the drain **1118** of a dual P-channel enhancement MOS-FET **1117**. The source **1120** of **1117** is coupled to the, termination power line **1121** (which carries +5 volts) of the UUT (pin **685** on connector **660**). Gate **1119** of MOS-FET **1117** is coupled to signal line **1060** coupled to a register in base unit **190** received from for activating SCSI bus termination. When this bit is set to a logic 0, a negative voltage is created at gate **1119** of **1117** effectively sourcing +5 volts from pin **685** (termination power) to **1116** on **1115**. **1115** steps down the voltage to 3.8 volts which is supplied to diode packs **1111**, **1112**, and **1113**. Diode packs **1111**, **1112**, and **1113** drop the voltage down another one volt to provide 2.8 volts to resistor pack **1110**. The 2.8 volts with 250 mA limiting resistors is provided to the remaining SCSI lines **1100** and meets the proper voltage and current specifications for SCSI termination of the bus.

On the other hand, when a logic "1" is present on signal line **1060**, the voltage at gate **1119** of **1117** is equalized, and thus removing SCSI bus termination from SCSI signal lines **1100**. No voltage is provided from drain **1118** of device **1117** to **1116**. Each of the

resistors in resistor pack **1110** appears as an individual open circuit to SCSI connector **660**. SCSI termination is thus effectively removed. In summary, an active signal over line **1060** causes **1010** to deactivate SCSI bus termination on lines **1100**, and a logic 0 received over line **1060** causes **1010** to activate SCSI bus termination on lines **1100**.

Testing UUT's

The testing of UUT's coupled to diagnostic tool **100** via port pack **160** in the preferred embodiment is accomplished via a sequence of routines embedded in ROM packs **140** and **150** which perform certain tests embedded in those ROM packs, or causes accesses to built-in diagnostics contained within the UUT. In the Macintosh family of personal computers, a series of routines known as the "Test Manager" is available in the system ROM of each computer. On a system which is not even able to power up and complete bootstrap initialization, this program will be unavailable. Therefore, certain "lowlevel" tests perform certain basic diagnostics such as testing a "power on" battery on the motherboard, testing voltage levels on certain signal lines, determining whether there is SCSI termination, or measuring the voltage of SCSI termination. A summary of these tests is set forth below. Each of these low level tests is written in 68HC11 assembly language for CPU **901** in the preferred embodiment but may be written in high level languages such as the "C" or Pascal programming language in alternative embodiments. These tests are perforated without entry into the diagnostic program by measuring voltages available on the various connectors, etc. which are coupled to tool **100** and a UUT. These tests are as follows:

Tests that Do Not Require the Test Manager to Function

Power Supply (PowrS)—This test measures the voltage at the power supply of the UUT. The value can range from 0.0 volts to 5.5 volts. This test runs on UUT's in the Macintosh brand of personal computers. This test has no pass/fail. It displays the voltage with a message stating what range of values indicates a functional power supply (usually >4.5 volts). This test requires a cable to be connected to a UUT and either port **610** or **620**.

Battery Voltage (Batt.)—This test measure the battery voltage off the ADB cable. The value can range from 0.0 volts to 10.0 volts. This test runs on Macintosh II motherboard and IIx brand computers. This test has no pass/fail. It displays the voltage of the battery with a message stating what range values indicates a functional power supply (usually >6.5 volts). This test requires a cable to be connected from a UUT to either port **610** or **620**.

Power Up Voltage (PwUpV)—The Macintosh IIcx brand computer uses a trickle current from the power supply to provide power up voltage (which is provided by a battery on the motherboard). This test measures the trickle current off a cable coupled to **610** or **620**. The value can range from 0.0 volts to 10.0 volts. It is very similar to battery voltage. This test has not pass/fail. It displays the voltage with a message stating what range of values indicates a functional power supply (usually >4.5 volts). This test requires a cable to be coupled to a UUT and either port **610** or **620**.

Power On CPU—This function, by imitating the action of pressing the soft power on key activates power in the UUT. This function runs on Macintosh II, IIx and IIcx brand personal computers. This function has no pass/fail. This test requires one cable to be connected from a UUT to port **610** or **620**.

SCSI Termination Check—This test checks the SCSI lines to see if a good termination exists on the bus. This test runs on all Macintosh brand personal computers. This test has pass/fail only. This test requires a SCSI cable to be connected from port **660** to the UUT.

SCSI Termination Power—This test measures the termination voltage off the SCSI bus. The value can range from 0.0 volts to 5.0 volts. This test runs on all members of the Macintosh brand personal computer family although may also run on other computers having a SCSI connector similar to **600**. This test has no pass/fail. It displays the termination voltage with a message stating what range of values indicates a functional terminator (usually >4.5 volts). This test requires a SCSI cable to be connected from connector **660** to the UUT.

Termination [On/Off]—This function turns on (or off depending on the previous state) the internal termination power of the SCSI bus in diagnostic tool **100**. This function runs on all members of the Macintosh brand personal computer family. This function has no pass/fail. It displays the termination condition as on or off on the screen. This function requires the SCSI cable to be connected to connector **660** and the UUT.

SCSI Reset—This function forces a hard SCSI reset on the bus. This function runs on all SCSI type hard drives. This function has no pass/fail. This function requires a SCSI cable to be connected to connector **660** and the drive.

SCSI Bus Scan—This test scans the SCSI bus looking for hard drives. It builds a table of any drives it finds and allows the user to obtain information about each drive such as drive type manufacturer and size. This test runs on all SCSI type hard drives. This test has no pass/fail. This test requires the SCSI cable to be connected to connector **660** and the SCSI bus being tested.

Test Manager Entry ("TstMd")—This function enters the Test Manager on the UUT. It cycles power and uses the SCSI bus to jump into the Test Manager on the UUT. This function runs on all machines equipped with a SCSI chip and having a Test Manager diagnostic program. This function fails only if it is not successful in entering the Test Manager. This function requires that cables be connected from the UUT to **660**, **610** or **620**, and a serial cable for the modem to be connected to port **630** or **640**. The process of entering the Test Manager is discussed in more detail with reference to FIGS. **12***a* through **12***d*.

Entering the Test Manager Diagnostic System

If the system is able to receive system power, however, the Test Manager diagnostic program may be entered in the UUT to perform certain tests at the user's choosing using diagnostic tool **100**. There are two ways in which the Test Manager is entered in a Macintosh brand personal computer using the preferred embodiment: by manually initiating a "non-maskable interrupt" (NMI) switch **620** on a Macintosh brand personal computer; or by causing the UUT to enter the Test Manager by simulating a SCSI device such as a disk drive and jumping to the Test Manager. The former option is difficult because the NMI switch must be issued within five to ten seconds after the UUT commences a bootup. Fairly tight timing constraints must be adhered to in

order to enter the Test Manager in this manner. In UUT's which are able to initiate system power and perform bootstrap initialization from an attached SCSI device, the latter method to enter the Test Manager is preferred. This is discussed in more detail with reference to FIGS. **12** through **15**.

FIGS. **12***a* through **12***d* show one process **1200** which checks certain basic conditions in the UUT and attempts to enter the Test Manager. Prior to bootstrap initialization from the simulated SCSI device provided by device **100**, certain basic conditions need to be established and tested for. Various corrective measures are taken to try to enter the Test Manager if entry is not successful on the first attempt. This process is shown in FIGS. **12***a* through **12***d*. When a UUT is connected to tool **100** for performing diagnostics, process **1200** is executed by tool **100** in order to attempt to enter the Test Manager. Process **1200** starts at step **1201**, as shown in FIG. **12***a*, by displaying to the user at step **1202** that power-up routine is being initiated in the UUT. At step **1203**, the SCSI bus termination and boot attempt counters, which are used for multiple attempts to terminate the SCSI bus or attempt to initialize the system, are cleared. The serial modem port **430** on tool **100** is initialized at step **1204** in order to prepare the unit for communication with a UUT. At step **1205**, it is determined whether the UUT is already powered up. If power is present in the UUT, then process **1200** proceeds to step **1221** as shown in FIG. **12***b*. It is determined whether the UUT is powered up, in a Macintosh brand computer system, using techniques well-known to those skilled in the art by sampling certain lines coupled to either ports **610** or **620** of peripheral port pack **160**. If the UUT is not powered up, as determined at step **1205**, then process **1200** proceeds to step **1206** wherein a power-up is attempted on the UUT using the sequence issued over port **610** or **620** on port pack **160** for certain models of the Macintosh brand computer. Again, it is determined at step **1207** whether the UUT has system power, and if it does, then process **1200** proceeds to step **1212** on FIG. **12***b*. If not, as determined at step **1207**, the message "Waiting for power on" is displayed at step **1208**, and process **1200** in FIG. **12***a* proceeds. It is then determined, at step **1209**, whether the "Back" key has been depressed by the user. This key allows the user to abort from any process currently executing in tool **100**. This is used if the user wishes to escape out of the waiting for power on loop of steps. **1209** and **1210**. If the "back" key is depressed, then process **1200** returns from Test Manager entry sequence **1200** at step **1211**. If the "back" key has not been depressed as determined at step **1209**, then the UUT is tested again at step **1210** to determine whether system power is present. Steps **1209** and **1210** are repetitively performed until it is detected that the UUT has powered up as determined at step **1210**. Once power is present, step **1210** proceeds to step **1212** in FIG. **12***b*.

As shown in FIG. **12**b, process **1200** continues at step **1212** wherein it is determined whether SCSI bus termination is present in the UUT. If SCSI bus termination is present, then **1212** branches to step **1218** to continue the initialization. If SCSI bus termination is not present, then process **1200** proceeds to step **1213**. **1213** determines whether this is the second attempt to terminate the SCSI bus, and if it is, step **1213** proceeds to step **1214** wherein the message "Cannot terminate SCSI (cable problem?), Try manual power on" is displayed to the user. Then, process **1200** branches to the exit process at step **1228** shown in FIG. **12***d*, and process **1200** is returned from at step **1235**. If, however, this is not the second attempt to terminate the SCSI bus, then step **1213** proceeds to step **1215** wherein the user is prompted with the message that "SCSI termination is missing," and the user is given the option to either "1) Terminate the bus or 2) cancel" the present operation. It is determined at step **1216** which option the user selected. If "cancel" is selected, step **1216** proceeds to step **1235** on FIG. **12***d* and process **1200** is returned from. If, however, the "cancel" selection was not made, as determined at step **1216**, then SCSI termination is attempted to be activated at step **1217** as shown in FIG. **12***b*. SCSI termination is performed, in the manner as discussed earlier, using soft SCSI termination circuitry **1010** as shown in FIG. **11**. Then, the message "Waiting for SCSI block read" is displayed at step **1218**, wherein tool **100** waits until the UUT attempts to read from the simulated SCSI device over port **660**. Tool **100** will cause the UUT to enter the Test Manager by "feeding" blocks to the UUT at step **1219**. This is discussed in more detail with reference to FIG. **13**. Tool **100** then waits three seconds in order for the Test Manager to be entered at step **1220** after the issuance of the entry sequence at step **1219**. It is determined at step **1221**, whether the UUT is in the Test Manager. If the UUT is not in the Test Manager, then process **1200** proceeds to step **1222** in FIG. **12***c*. If, however, the UUT was in the Test Manager, then process **1200** proceeds to step **1224** in FIG. **12***d*.

As shown in FIG. **12***c*, process **1200** continues at step **1222** wherein it is determined whether UUT power was on already, or whether the power-on sequence at step **1206** had to be performed. If power was already on in the UUT, then process **1200** proceeds to step **1229** wherein it is ascertained whether this is the second attempt to boot the Test Manager in the UUT. If it is the second attempt to boot, then the message "SCSI/serial problem. Try manual entry (press BACK)" is displayed at step **1223**. If process **1200** activated power in the UUT, then step **1222** also proceeds to step **1223** to display the message. After the message if step **1223** is displayed, the exit process at step **1228** shown in FIG. **12***d* is performed, and process **1200** is returned from at step **1235**. If this is not the second boot attempt as determined at step **1229**, then the message "Waiting for power off" is displayed at step **1230**, and the UUT is checked via ports **610** and **620** to see whether power is off in the UUT at step **1231**. If it is not, and the "back" key was not depressed (causing interruption of process **1200**), then steps **1231** and **1232** are performed repetitively until it is determined at step **1231** that power is off in the system, or the "back" key is depressed. When power-off is detected in the UUT at step **1231**, then process **1200** proceeds back to step **1206** to issue a "Power-on" signal to the UUT as shown in FIG. **12***a*. If the "back" key is depressed, as determined at step **1232** (interrupting Test Manager Entry Process **1200**), then process **120**)is returned from at step **1235** shown in FIG. **12***d*.

If the UUT was not in the Test Manager as determined at step **1221** in FIG. **12***b*, then process **1200** proceeds to step **1224** in FIG. **12***d*. Step **1224** in FIG. **12***d* checks the UUT's tinting. This is determined using techniques well-known to those skilled in the art by sampling various signals received over the ports on port pack connector **160** to see if the UUT is responding within defined tolerances. If the timing is bad as deter-

mined at step **1225**, then process **1200** proceeds to step **1226**. If the timing is not bad as determined at step **1225**, then the message "Communications established: CPU is in Test Manager mode" is displayed at step **1233**, and tool **100** waits an additional five seconds for a user intervening keypress, at step **1234**. If no keypress is received, process **1200** returns at step **1235**. If, however, the timing was bad as determined at step **1225**, then process **1200** proceeds to **1226** to determine whether tool **100** activated power in the UUT. If power was on already (tool **100** didn't activate the power), then a "RESET" command is issued to the UUT at step **1236** and process **1200** continues at step **1221** as shown in FIG. **12***b*. If power was activated in the UUT by tool **100**, then the message "Test Manager OK but timing problem found" is displayed at step **1227**, and the exit process is branched to at step **1228**. Then, process **1200** is returned from at step **1235**.

Thus, at the end of process **1200**, the UUT should be in its Test Manager mode, or the user is prompted with various options that he may take in order to perform diagnostics on the UUT. The entry into the Test Manager by simulating a SCSI device mentioned with reference to step **1219** will now be discussed.

Entry Into the Test Manager by Simulating a SCSI Device

Entry into the Test Manager is performed by diagnostic tool **100** by simulating a SCSI device to the UUT. This is accomplished by sensing signals received through connector **660** transmitted by the UUT. When the appropriate SCSI initialization signals are received through port pack **160** from a UUT coupled to connector **660**, diagnostic tool **100** issues response signals through connector **660** to the UUT to cause the system to jump to the Test Manager. It performs this by simulating a SCSI device such as a disk drive using a driver descriptor map (DDM) and partition map entry which is expected by certain UUT's using SCSI devices. These entries are shown in more detail in FIGS. **14** and **15**. As is well-known to those skilled in the art, a computer system having a SCSI interface, such as a UUT probes the SCSI bus to determine if SCSI devices are present. Each SCSI device identifies itself with an identification number. This is known as the arbitration phase. Once ID's have been sampled by the UUT, selection of a device may be accomplished. This is known as the selection phase. After arbitration and selection, the UUT can issue a command (such as a read or a write) and the device will respond with data. The preferred embodiment senses the issuance of the arbitration and selection signals, and simulates a SCSI device with an ID=6, which is the default highest priority SCSI ID number besides the UUT in a Macintosh brand computer. Other responses to probes by the UUT are now discussed. These are performed, as is well-known to those skilled in the art, by sensing the arbitration, selection, and command signals sent by the UUT, and device **100** responds in a manner expected from a SCSI device. The signals issued to the UUT will now be discussed.

The SCSI Test Manager entry process performed by the UUT is shown as **1300** in FIG. **13** (referred to in step **1219** of process **120**)). Process **1300** is performed by a UUT when allowed to continue to perform bootstrap initialization and has a SCSI device coupled to one of its ports through port **660**. Process **1300** starts at step **1301** wherein, at step **1302**, a first available SCSI device on the bus is selected that has an ID=6 (this is the highest priority SCSI device in a Macintosh brand computer). When the UUT requests an access to the SCSI device having ID=6, diagnostic base unit **190** senses this signal and transmits responsive data to the UUT via port pack **160**. The fore, at of data responsive to requests issued by the UUT is shown in FIG. **14** and is shown in more detail in FIGS. **14** and **15**. At step **1303**, the UUT will attempt to read the first 256 bytes of block **0** of this simulated device by issuing read commands through port **660** to tool **100**. Block **0 1401** is shown with reference to FIG. **14**. This contains the driver descriptor map (DDM) for the simulated SCSI partition. The UUT attempts, at step **1304**, to look for a driver descriptor map such as **1401** with a value equaling **$4552** (this indicates that the SCSI device has been formatted). This is indicated as field **1501** in FIG. **15**. Upon receiving the address for the first 256 bytes of block **0** of the unit with ID=6, diagnostic tool **100** transmits DDM **1401** onto the SCSI bus. Driver descriptor map **1401** comprises two portions, **1401** a and **1401***b*. **1401***a* contains "real data" which resides in non-volatile memory of tool **100** and is transmitted to the requesting UUT. The remaining portion **1401***b* of the data is zero-filled and transmitted to the UUT through port **660** byte-by-byte from a register in tool **100**. The remaining blocks shown in FIG. **14**: **1402***b*, **1403***b*, **1404***b*, and **1405***b* are similarly zero-filled and transmitted to the UUT. The remaining "real" portions of driver descriptor map **1401***a* is shown in FIG. **15**. Driver descriptor map **1401***a* indicates the size in blocks of the device in field **1502** (a one word field) and the number of blocks on the device in field **1503** (a 32-bit longword). Blocks are 512 bytes long, in a preferred embodiment. Field **1504** indicates the device type (a one for disk drive), **1505** has the device ID type (one). Field **1506** also contains a one, in the preferred embodiment, a value expected by Macintosh brand UUT's. The number of driver descriptors is contained in field **1507**. In this example, there is only one. The driver descriptor starts at field **1508**. The first block of the driver resides at block four as indicated by field **1508** (a 32-bit longword), and the driver size is exactly one block long as indicated by field **1509**. Lastly, the driver type is of type **1**, which is contained in field **1510**, in the Macintosh brand family of personal computers. After the driver descriptor map is read at step **1304**, step **1305** reads the pseudo device driver of the simulated SCSI device at block **4**, shown as **1405** in FIG. **14**. The format of blocks **1402** through **1405** are discussed with reference to FIG. **16**.

Partition map entries, such as **1402**, **1403**, **1404**, or **1405** are one-block entries containing information about the SCSI partitions which are being accessed. The pseudo device driver of the preferred embodiment resides in one of these partition map entries, and is shown as **1405** in FIG. **14**. Each of the partition map entries has a specific format, which is well-known to those skilled in the art, and is set forth in *Inside Macintosh*, Volume V available from Addison-Wesley at pages V-576 through V-582. Because the actual data contained within fields in blocks such as **1402** through **1405** is well-known to those skilled in the art, a detailed discussion of that will not be set forth here. Each of the blocks **1402** through **1405** contains a first portion such as **1402***a* through **1405***a*, which contains real data in order to conserve space in the ROM of the preferred embodiment. The remaining portions such as **1402***b* through **1405***b* contain dummy data which is transferred to the UUT when request for the remainder of the block is made by the

UUT. In each of the partition map entries **1402** through **1405**, the actual portion read in by the UUT, **1402***a* through **1405***a*, is 64 bytes in length. The remaining portions **1402***b* through **1405***b* are padded with zeros. The partition map entries are then read by the UUT. At step **1306**, the partition map **1402** for tile operating system is read. Once this is done, the partition map entry for the pseudo partition is read at step **1307**. Then, the partition map entry **1404** for tile driver is read in at step **1308**. This gives the system the address at which to start to execute the driver contained in **1405**. Then, at step **1309**, the partition map entry for the operating system **1402** is read again. At step **1310**, the driver residing at the location indicated by **1404** is called to install itself and the driver causes a jump to the Test Manager to occur. Once process **1300** is complete, at step **1311**, the Test Manager is running in the UUT, which can now be accessed and communicated with by diagnostic tool **100**.

In summary, diagnostic tool **100**, in one embodiment of the invention, simulates a SCSI device with an ID = 6 for a Macintosh brand personal computer. A series of requests is made to tile device to read various information off a simulated SCSI device. Blocks are returned from this simulated device as shown in FIG. **14** and described with reference to FIG. **13**, and tile diagnostic device **100** returns blocks which are expected by the UUT. Blocks are fed to the UUT in tile following order: **1401**, **1405**, **1402**, **1403**, and **1402**. As a last step, the device driver residing in block **1405** is called to execute itself thus causing a jump to occur to the Test Manager which resides in the UUT system ROM.

Once entry into tile Test Manager of tile UUT has been perforated, various tests within the Test Manager may be accessed and executed. It can be appreciated by one skilled in the art that other architectures possessing similar test routines may be accessed in a similar manner. This will allow entry and testing in an automated fashion by diagnostic apparatus **100**. The commands set forth in Table **4** are provided for execution of the Test Manager remotely via a coupling with the UUT to serial port **640** on port pack **160**. Each command is preceded by a "*" which indicates that a command is following on serial port **640**. When a UUT is activated and the Test Manager is entered, in one embodiment, the serial port of the UUT defaults to a **9600** baud configuration and expects "*" commands to be received via serial port **640** coupled to the serial port of the UUT. Responses to tests requested by diagnostic apparatus **100** are also provided across the serial port. Specific commands are summarized as follows:

TABLE 4

Initiating Command	UUT Response	Command Name	Description
*S	*S	Stop initial input message and Start UUT ROM Monitor	This command is the "handshake" that initializes the UUT ROM monitor to accept test commands. It also silences the UUT (if the UUT sends out a repeating error message upon failure or if the ROM monitor has been invoked).
*L xx xx xx xx	*L	Load address	This command is followed by 4 bytes that specify the address where the UUT will start loading (or downloading) data into RAM. Response indicates that the UUT received the valid command (handshake). Can be used to load tests into the UUT.
*B xx xx	*B	Byte count	This command is followed by 2 bytes that specify the number of bytes to be downloaded. (*L must precede this command). Response indicates UUT received the valid command (handshake).
*D xx	*D	Download data	This command is followed by xx xx (2 hex) bytes as specified by the *B command. (*L & *B must precede this command). Response indicates UUT received valid command (handshake).
*C xx xx xx xx	*C	Checksum data	This command requests memory range checksum specified by starting location and number of bytes (via *L and *B respectively). Returns checksum followed by *C. Must be used with *L and *B.
*G xx xx xx xx	*G	Go . . . (execute)	This command is followed by 4 hex bytes that specify the address where the UUT will start running a program. (To run a downloaded program, *L, *B, and *D must precede this command). Response indicates UUT received the valid command (handshake).
*Oxx xx xx xx	*0	O = low or bottom RAM test address	This command is followed by 4 hex bytes that specify the address where the UUT will start running a RAM test. (A pointer). Response indicates UUT received the valid command (handshake).
*1 xx xx	*1	1 = high or top	This command is followed by 4 hex bytes that specify the address where the UUT will start running a RAM test. (A pointer). Response indicates UUT received the valid command (handshake).
*2 rr	*2	Read CPU register	Requests CPU register data specified by rr. Response returns 32-bit register information from the UUT CPU register, rr. The

TABLE 4-continued

Initiating Command	UUT Response	Command Name	Description
			register dump is followed by *2 to indicate the end of transmission.
*3 rr xx xx xx xx	*3	Write CPU register Registers: D0–D7 SP A0–A6 SR	Writes 32-bit value to register specified by rr. Example: *3D301F9AB35*3 *2SP_____000E952*2 Underlined section returned by Test Manager)
*4	*4	Clear UUT error register. (e.g., D6.L & D7.W)	Clears the error storage registers on the unit under test. Response indicates UUT received the valid command (handshake).
*5	*5	Re-enter ROM monitor	Re-starts ROM monitor entry and re-calls generic header (e.g., *APPLE*xxxxxxxxxxx*1*). Response indicates UUT received the valid command (handshake).
*A	*A	ASCII mode	Sets ASCII character mode. Sends and receives ASCII data. Response indicates UUT received the valid command (handshake).
*H	*H	Hex mode	Sets hex character mode. Sends and receives hex data. Response indicates UUt receives the valid command (handshake).
*R xx xx xx xx xx xx	*R	Result request	Requests current status or test results. Response returns a 12-character result code that specifies the current error status.
*M xx . . .	*M	Memory dump	Requests memory data specified by *L and *B (*L and *B must precede this command). Response returns memory information as specified by the *L and *B commands. The memory dump is followed by *M to indicate the end of transmission.
*E	*E	Echo	Requests the UUT to echo externally transmitted characters back to the external device. Response indicates UUT received the valid command (handshake).
*I	*1	Initialize	Requests the UUT to re-initialize itself. This terminates serial communications. (Re-boot).
*T tt tt pp pp oo oo	*T	ROM-resident Test request	Requests the UUT ROM test to be run. This command is followed by 8 hex bytes; "u u" is the "pass count" (or number of times test is to be run); "oo oo oo oo" is the option word. Response indicates UUT received the valid command (handshake). Examples of ROM Test number (tt tt) activities: 0000 - Size Memory 0001 - DataBus Test 0002 - Mod3Test 0003 - Addrline Test 0004 - ROMTest 0005 - Rev3ModTest 0006 - StartUpROMTest

An additional feature provided by the Test Manager is the ability to download specific tests into the Test Manager. This is performed using the "*D" command and such tests may be executed by typing in the "*G" command with the address of the test loaded using the "*D" command. The last command in any user-specified test using "*D" must jump back to the starting address of the Test Manager to provide the appropriate response to diagnostic tool **100**. It can be appreciated by one skilled in the art that a variety of entries into system-specific test functions such as the ones provided above may be performed.

Tests Requiring the Test Manager

Most of tile tests run ell diagnostic tool **100** require the UUT to be in the Test Manager. This requires the machine to have a limited amount of functionality. Most of these tests download a piece of code into the UUT to run in the native environment. If the user attempts to run one of these tests without being in the Test Manager, he will be given a prompt to enter the Test Manager by diagnostic tool **100**. In order to enter the Test Manager automatically, each test requires that the UUT be coupled to ports **660** and **610** or **620** using appropriate cables. Also, tool **100** must be coupled to the UUT through port **630** using a serial cable to communicate with the Test Manager.

ROM Checksum (ROMck)—This test checksums all of the ROM's in the CPU (if more than one exists). This test will not identify individual ROM failures. This test fails if the checksum does not match the one stored in the ROM, otherwise it passes.

RAM Size (RAMsz)—This test determines the size of the RAM in the CPU. The test finds the largest SIMM in the bank and uses that to determine the size. For example, if the test finds three 256K SIMM's and one 1 Meg SIMM, it will size the machine to 4 Megs. This test has no pass or fail. It reports the size of memory it finds and indicates if it thinks the memory is misconfigured (a system where the largest SIMM's are in bank A instead on bank B) or bad (which may just be mismatched (a system which has different size SIMM's in the same bank)).

Address Test (Addr.)—This test checks the address lines of the CPU. It determines if the lines are functioning properly. This test has pass or fail only.

Databus Test (Data)—This test checks the data lines of the CPU. It determines if the lines are functioning properly. This test graphically shows catastrophic errors and missing SIMM's.

SIMM's Test (SIMMs)—This test uses the results from RAM Size to read and write RAM to determine if it is good. This test graphically shows catastrophic errors, missing SIMM's and individual bad SIMM's.

VIA Test (VIA)—This test checks the VIA's (Versatile Interface Adaptors or Peripheral Controller Chips) for functionality. This test gives pass or fail only.

Clock Test (Clock)—This test checks the UUT real time clock chip for functionality. This test gives pass or fail only.

Parameter RAM Test (P/RAM)—This test verifies that the PRAM (parameter RAM which controls data, and mouse information) can be read from and written to. This test gives pass or fail only. NOTE: There are two modes of operation for this test. One mode saves and restores the contents of PRAM (parameter RAM of the UUT), and the other mode clears out all of PRAM.

SCC Test (SCC)—This test verifies that the SCC chip functions in all modes and that the serial bus can send and receive data correctly. This test gives pass or fail only.

SCSI Test (SCSI)—This test verifies that the SCSI (small computer system interface) chip functions in all modes and that the SCSI bus can send and receive data correctly. This test gives pass or fail only.

SWIM/IWM Test (SWIM)—This test determines if the CPU has an IWM or a SWIM (floppy disk controller chips) connected and then tries to initialize the chip. This test gives pass or fail only. This test can only detect catastrophic failures in the IWM/SWIM. The floppy drive tests provide a more comprehensive test of the chip.

FPU Test (FPU)—This test verifies that the FPU chip (floating point math coprocessor) functions. This test runs on Macintosh II, IIx, IIcx brand family of personal computers. This test gives pass or fail only.

Sound Test (Sound)—This test verifies that the sound chip registers function and that data can be read from and written to the chip. This test also measure the sound out for volume and frequency. This test gives pass or fail only.

ADB Test (ADB)—This test verifies that the ADB (mouse and keyboard) transceivers, the portions of the VIA that control ADB, the ADB line and the ADB port is functional. This test gives pass or fail only. This test requires that both ADB cables be connected from the UUT to ports 610 and 620.

Video Card Test (Video)—This test determines what video cards are in what slots and builds a menu for the user. The user can select which card he wants to test. The test will then test the RAM on the selected video card. This test runs on the Macintosh II, IIx, and IIcx brand family of personal computers. This test gives graphic representation of which SIMM's are bad or fails the card if the soldered SIMM's are defective.

Floppy Drive Test (Drive)—This test determines what floppy drives are in what ports and builds a menu for the user. The user can select which drive he wants to test. It then tests the ability of the drive to read, write, and seek. This test gives pass or fail only.

Power Off CPU—This test shuts off power to the CPU. This test funs on Macintosh II, IIx, and IIcx brand family of personal computers.

Thus, an invention for diagnosis for computer systems has been described. Although the present invention has been described particularly with reference to FIGS. 1 through 15, it will be apparent to one skilled in the an, however, that the present invention has utility far exceeding that disclosed in the figures. It is contemplated that many changes and modifications may be made, by one of ordinary skill in the art, without departing from the spirit and scope of the present invention as disclosed above.

What is claimed is:

1. A device for terminating a bus comprising:
   a. first means for activating termination of the bus;
   b. second means coupled to the first means for supplying power, the second means supplying power upon activation of the first means;
   c. third means coupled to the second means for limiting voltage received from the second means; and
   d. fourth means coupled to the third means for limiting transient voltages on the bus.

2. The device of claim 1 wherein the first means comprises a flag.

3. The device of claim 1 wherein tile second means comprises a MOS-FET wherein the source of the MOS-FET coupled to a power supply, drain is coupled to the third means, and tile gate is coupled to the first means.

4. The device of claim 1 wherein the third means comprises a low dropout voltage regulator.

5. The device of claim 1 wherein the fourth means comprises at least one diode.

* * * * *

United States Patent [19]

Keener et al.

[11] **Patent Number: 5,033,049**

[45] **Date of Patent: Jul. 16, 1991**

[54] **ON-BOARD DIAGNOSTIC SUB-SYSTEM FOR SCSI INTERFACE**

[75] Inventors: **Don S. Keener**, Boca Raton; **Andrew B. McNeill**; **Kevin L. Schelern**, both of Deerfield Beach, all of Fla.

[73] Assignee: **International Business Machines Corporation**, Armonk, N.Y.

[21] Appl. No.: **364,363**

[22] Filed: **Jun. 12, 1989**

[51] **Int. Cl.**[5] **G06F 11/00**

[52] **U.S. Cl.** **371/23**; 364/200; 371/16.1

[58] **Field of Search** ..... 371/16.1, 16.2, 23; 364/200 MS File, 900 MS File

[56] **References Cited**

U.S. PATENT DOCUMENTS

3,889,109	6/1975	Blessin	235/153 AK
3,931,506	1/1976	Borrelli	235/153 AC
3,958,111	5/1976	Hackett	235/153 AK
4,275,464	6/1981	Schmidt	371/15
4,339,819	7/1982	Jacobson	371/16
4,385,349	5/1983	Ashford	364/184
4,499,580	2/1985	Takahashi	371/17
4,500,993	2/1985	Jacobson	371/16
4,535,456	8/1985	Bauer	371/16
4,638,455	1/1987	Yamazaki	364/900
4,669,004	5/1987	Moon	360/77
4,718,064	1/1988	Edwards	371/20
4,779,196	10/1988	Manga	364/200
4,783,705	11/1988	Moon	360/77
4,868,825	9/1989	Koeppe	371/23
4,905,184	2/1990	Giridhar et al.	364/900
4,914,656	4/1990	Dunphy, Jr. et al.	371/10.2

OTHER PUBLICATIONS

Aseo, "Disk Interfacing Gets Smart", *ESD*, Nov. 1987, pp. 77–84.

Ow-Wing, "Diagnostic Registers Simplify System Self-Testing", *Electronic Design*, Oct. 27, 1983, pp. 155–164.

Ow-Wing, "Strategy for Diagnostic Design", *New Electronics*, Oct. 18, 1983, pp. 74–77.

Robinson, "SCSI Interface Demands New Test Methods", *Mini-Micro Systems*, Apr. 1988, pp. 35–39.

Squires, "Outsmarting Smart Interfaces to Test Winchester Drives", *Electronics Test*, Feb. 1988, pp. 39–43.

"Solving the Test Problem in SCSI Disk Drives", *Electronics*, Feb. 17, 1986, pp. 35–37.

"Western Digital Slashes SCSI Bus Overhead Time", *Electronics*, Aug. 20, 1987, pp. 64–65.

Primary Examiner—Charles E. Atkinson

Attorney, Agent, or Firm—Pollock, Vande Sande & Priddy

[57] **ABSTRACT**

On board diagnostic capability for a SCSI controller is provided within an adapter by providing a gate array driven by a microprocessor on board the adapter. The gate array has data and control inputs driven from the microprocessor and data and control outputs which are dot OR'ed with corresponding in/out terminals of the SCSI controller. A reset signal from a SCSI bus forms a further input to the gate array. For testing purposes the microprocessor drives the gate array inputs to simulate a fault-free or faulty device. The microprocessor detects the response of the SCSI controller to the device simulation and thereby can determine the state of health of the SCSI controller.

9 Claims, 3 Drawing Sheets

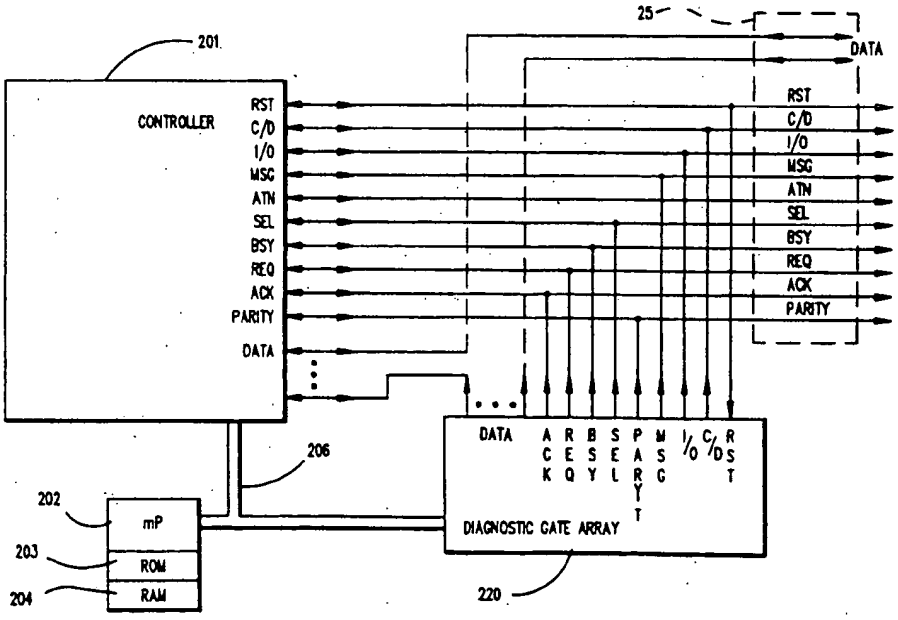


FIG. 1

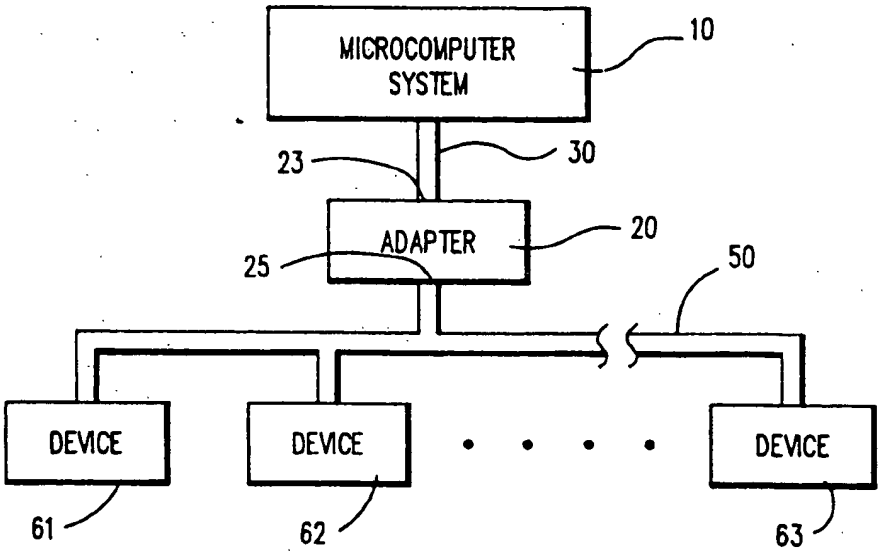
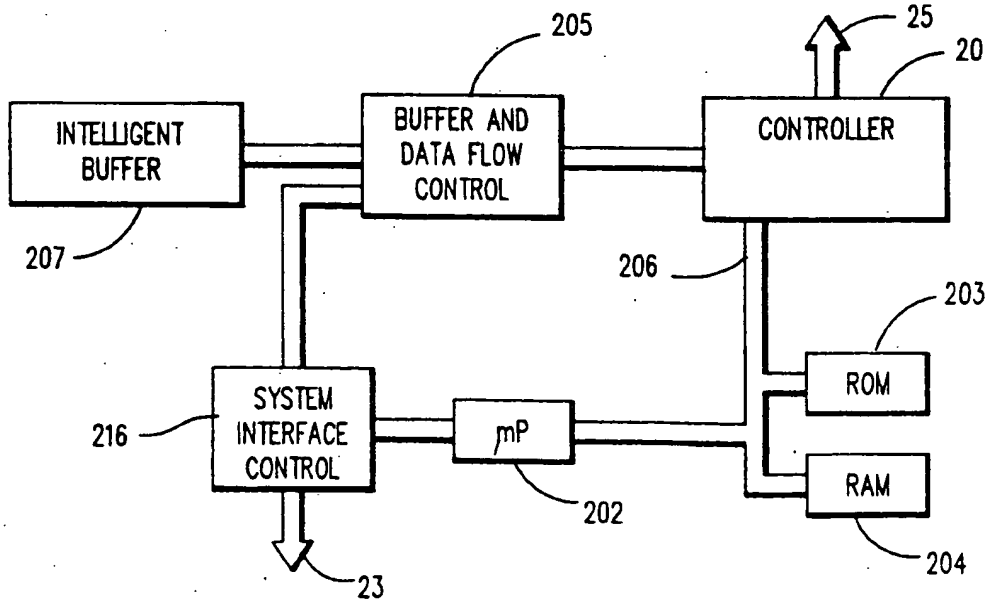
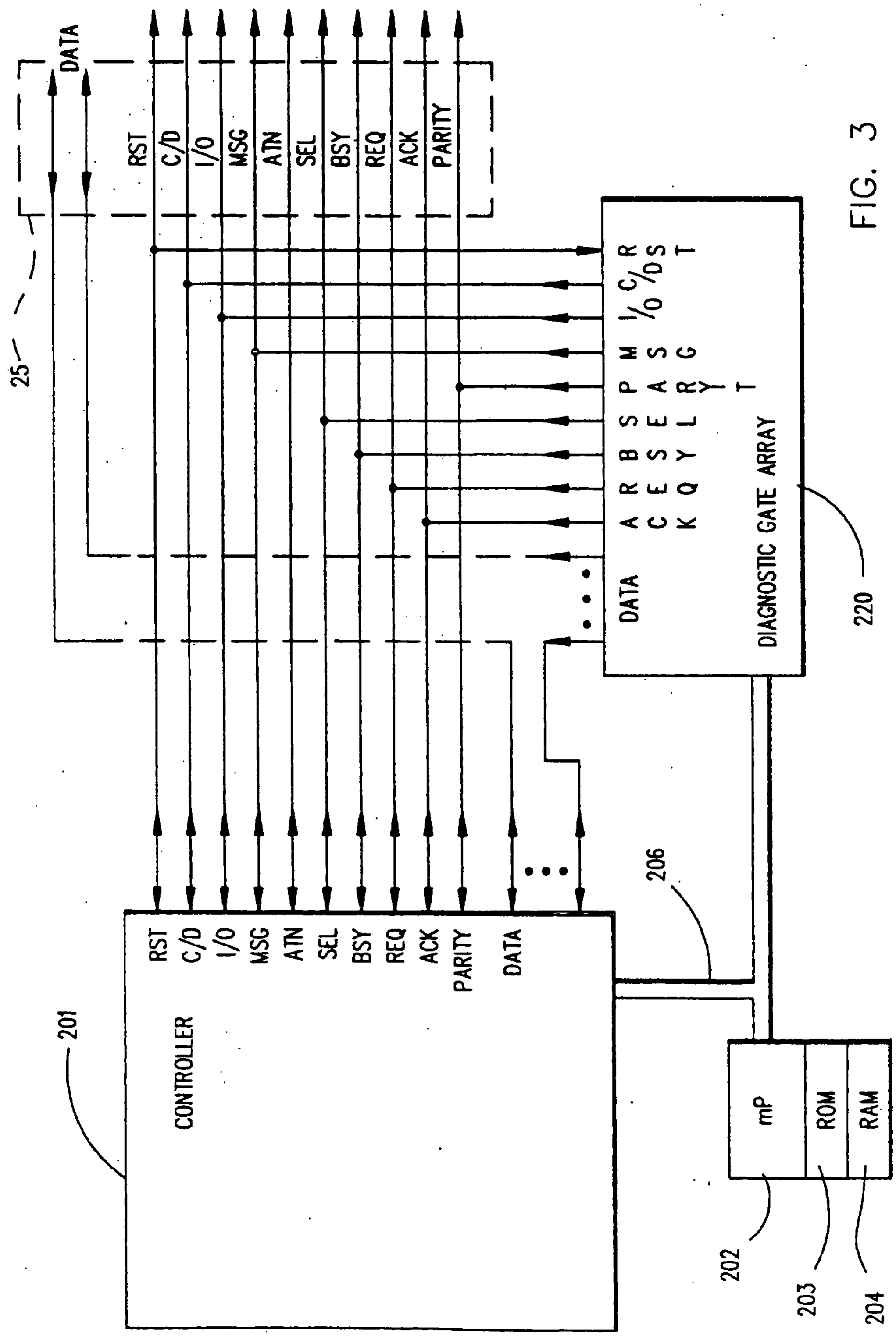

MICROCOMPUTER SYSTEM
10
23
30
ADAPTER
20
25
50
DEVICE
DEVICE
DEVICE
61
62
63

FIG. 2


205
25
20
INTELLIGENT BUFFER
BUFFER AND DATA FLOW CONTROL
CONTROLLER
207
206
203
ROM
216
SYSTEM INTERFACE CONTROL
mP
RAM
202
204
23



DATA
RST
C/D
I/O
MSG
ATN
SEL
BSY
REQ
ACK
PARITY
25
DIAGNOSTIC GATE ARRAY
220
CONTROLLER
201
206
mP
ROM
RAM
202
203
204

FIG. 3

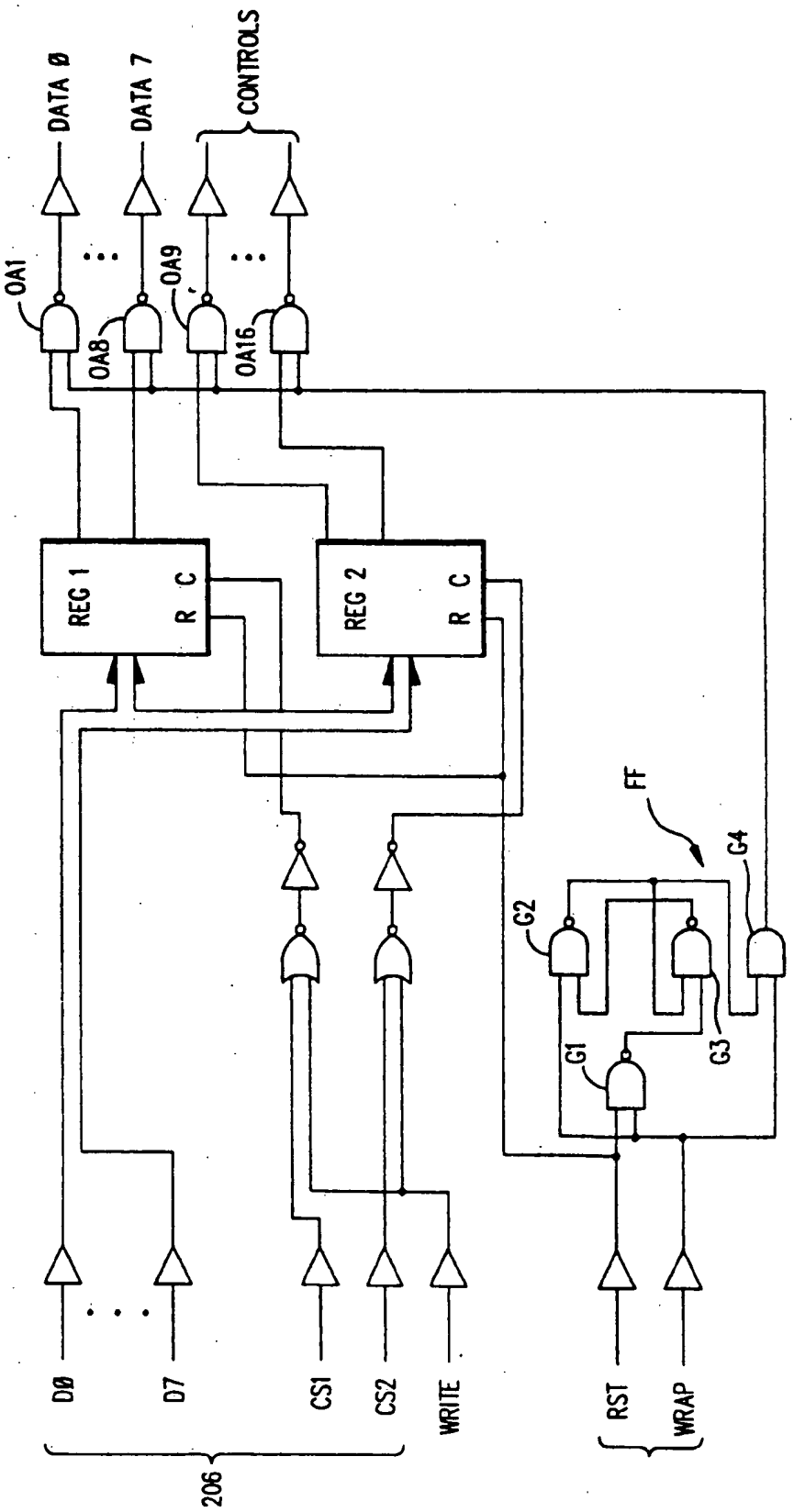

DATA Ø
DATA 7
CONTROLS
OA1
OA8
OA9
OA16
REG 1
R C
REG 2
R C
FF
G4
G2
G1
G3
DØ
D7
CS1
CS2
WRITE
RST
WRAP
206

FIG. 4

ON-BOARD DIAGNOSTIC SUB-SYSTEM FOR SCSI INTERFACE

FIELD OF THE INVENTION

The invention relates to testing devices complying with the Small Computer System Interface (SCSI) and more particularly to a system with on board testing and diagnostic capability for devices such as a SCSI controller.

BACKGROUND OF THE INVENTION

Diagnostic subsystems have, in the past, been applied in at least two different applications. In one application, diagnostic systems and sub-systems have been used as a part of the manufacturing process. More particularly, after a product is manufactured it is connected to a diagnostic system or sub-system to insure that the manufactured device meets manufacturing specifications and is in condition for sale. In an entirely different application, diagnostic sub-systems have been included as part of computer systems. Typically such an on board diagnostic sub-system is initiated into operation as part of Power On Reset (POR) sequence each time the computer system is powered up or reset. The diagnostic sub-system performs a routine of tests and either allows normal processing to be initiated if the tests are successfully passed or reports some information about any tests which are not successfully completed.

One problem faced by the diagnostic sub-system designer is typified by the expandable nature of the Personal Computer (PC). More particularly the nature and complement of devices included in the computer system cannot be predicted. This unpredictability limits the extent and nature of the diagnostics. Prior art I/O devices, for example hard files, are associated with a device controller, and the device controller is usually tested with the device connected to the controller. Prior art ST506 and Enhanced Small Device Interface (ESDI) devices are examples of this architecture.

More recently, with the advent of the SCSI standard, logic which is dedicated to operation of devices such as a hard file has been moved from the controller to the hard file assembly itself. This leaves a generic interface which actually operates as a low cost network and is capable of supporting hard files, diskette drives, CD ROM drives, printers etc. This new architecture requires a completely different method of testing the SCSI controller from those used in the ST506 and ESDI arrangements. See in this regard, "Solving the Test Problem in SCSI Disk Drives" appearing in *Electronics*, page 35 at seq., Feb. 17, 1986; Robinson, "SCSI Interface Demands New Test Methods", *Mini-Micro Systems*, page 35, April, 1988; Squires, "Outsmarting Smart Interfaces to Test Winchester Drives", *Electronics Test*, February, 1988; "Western Digital Slashes SCSI Bus Overhead Time" in *Electronics*, Aug. 20, 1987 and Aseo, "Disc Interfacing Gets Smart", *ESD: The Electronics System Design Magazine*, page 77, November 1987.

The SCSI or generic interface raises a number of testing issues especially with respect to the SCSI interface between a system, such as a Personal Computer system, and a plurality of devices coupled to the SCSI interface through a SCSI bus. Although in prior devices the hardware between a system and a peripherial device was identified as a controller this application will refer to the hardware interfacing the computer system and the SCSI devices as a SCSI adapter. Thus the SCSI adapter has a port coupled to a first bus which first bus is coupled to a port of the computer system. The SCSI adapter has a second port which is coupled to the SCSI bus and via that bus to one or more SCSI devices. Typically the SCSI adapter includes a SCSI controller as a component.

One problem raised by the SCSI interface is caused by the inability to assume a particular configuration of devices connected on the SCSI bus. Because no configuration can be assumed, it is not possible to select specific test procedures, for execution, based on the configuration. For example one configuration may require a collection of printers, another configuration may include a collection of read only CD ROMS, another configuration may have a plurality of read/write hard files, a different configuration may have all of the above and finally a further configuration may include devices still being developed. The inability to assume a particular device configuration, coupled with the limitations a particular device may provide in testing many of the modes of operation requires a new and effective test capability which is effectively independent of the devices connected on the SCSI bus.

A second problem is the necessity to isolate defects in the drivers and receivers in the SCSI adapter or more particularly, the SCSI controller contained in the adapter. Often if the drivers and receivers are defective, the defects cannot be isolated between the adapter, the cable or the device.

A third problem is the testing environment. The test sub-system may operate in an environment in which it is not the autonomous bus master, this is particularly true of the SCSI environment. While in the SCSI environment the adapter usually has the highest priority, nevertheless other devices may attain control of the bus and that condition must be respected; furthermore there may be more than a single adapter on the bus. As a result the diagnostic sub-system must take into account the presence of other devices which may be connected on the bus. If the diagnostic sub-system assumes control of the SCSI bus, it is conceivable that it can interfere with the operation of other bus users to the detriment of the entire system.

SUMMARY OF THE INVENTION

The invention solves the foregoing problems and provides an advance in the art as follows. In one form of the invention, the SCSI controller, component of the SCSI adapter, is coupled to the SCSI bus and more particularly to eight data conductors, a data parity conductor, a C/D (commands/data) conductor, a MSG (message) conductor, a I/O (input/output) conductor, a SEL (select) conductor, a BSY (busy) conductor, a REQ (request) conductor, an ACK (acknowledge) conductor, and a RST (reset) conductor. Dot OR'ed to the C/D, I/O, MSG, SEL, BSY, REQ, ACK and parity conductors of the SCSI controller are the outputs of a diagnostic gate array. The gate array has eight data output conductors which are dot OR'ed with the eight data conductors coupled to the SCSI controller data I/O terminals. The outputs of the gate array are derived from a plurality of registers contained in the diagnostic gate array. Inputs to the diagnostic gate array are provided by a dedicated micro-processor, dedicated to the adapter. The micro-processor output which is coupled to the diagnostic gate array includes eight data bits,

write select control signals and a wrap (test) control signal. A further input to the diagnostic gate array is a signal appearing on the RST conductor of the SCSI bus.

With the capability afforded by the diagnostic gate array, the SCSI controller can be exercised by diagnostic microcode and the outputs of the diagnostic gate array, also driven by the micro-processor can simulate the response of an attached device operating either in a fault-free or purposefully simulated fault mode, and generally in both modes. Diagnostic microcode which exercises the SCSI controller causes the SCSI controller to select itself thereby eliminating the possibility of corrupting the integrity of devices connected to the SCSI bus. Diagnostic microcode, coupled through the diagnostic gate array simulates a SCSI device attached to the SCSI bus to thereby test the arbitration, selection, re-selection and read/write functions of the SCSI controller. Testing can put the SCSI controller in both an initiator and target modes using DMA and automatic PIO methods where appropriate. The diagnostic microcode can also implement phase and parity testing.

The testing can be implemented with or without the SCSI bus connected to the SCSI connector on the adapter, so long as the connector is properly terminated. Because the input/output terminals of the SCSI controller are dot OR'ed to outputs of the diagnostic gate array, the presence of a SCSI bus and/or the presence of devices connected to the SCSI bus is completely immaterial to the diagnostic operation (again assuming proper termination to prevent signal reflections).

Accordingly, in one aspect the invention provides device adapter capable of controlling a variety of peripheral devices each designed to comply with a Small Computer System Interface (SCSI) standard, said device adapter including:

a connector for connection to a SCSI bus;

a micro-processor and dedicated memory coupled thereto;

a SCSI device controller coupled to and driven by said micro-processor, said SCSI device controller including dedicated input/output terminals for driving selected conductors of said SCSI bus and for responding to selected conductors of said SCSI bus, and

a gate array with a set of inputs coupled to said micro-processor and a set of outputs coupled in common with said input/output terminals of said SCSI device controller.

It should be apparent from the foregoing that the diagnostic gate array, in combination with other elements of the SCSI adapter, allows diagnostic operations either in the manufacturing process or as part of a POR sequence.

In the form of the invention already described the diagnostic gate array and the SCSI controller may be implemented as separate electronic chips. However in another (or second) form of the invention the diagnostic gate array can be integrated with the SCSI controller to form a single electronic chip. This single electronic chip, in accordance with the second form of the invention, may include the functions attributed to the SCSI controller and the functions attributed to the diagnostic gate array in accordance with the description of the first form of the invention.

BRIEF DESCRIPTION OF THE DRAWINGS

The invention will be further described in the following portions of this specification when taken in conjunction with the attached drawings in which:

FIG. 1 is a block diagram of a micro computer system including an adapter 20 its associated interface and a plurality of devices coupled to that interface;

FIG. 2 is a detailed block diagram of the adapter 20;

FIG. 3 is a further detail of the adapter 20 showing a typical SCSI controller 201 and a diagnostic gate array 220, in accordance with the present invention; and

FIG. 4 is a schematic of the diagnostic gate array 220.

DETAILED DESCRIPTION OF PREFERRED EMBODIMENTS

FIG. 1 is a block diagram of a typical microcomputer system 10 showing, in detail, an adapter 20 coupled to and controlling a plurality of devices 61–63 through a bus 50. The bus 50 is coupled to a port 25 on the adapter. In addition, the adapter 20 includes a port 23 through which the adapter is coupled to the bus 30. In a specific embodiment of the invention the adapter 20 comprises a SCSI adapter e.g., an adapter conforming to the SCSI standard. The adapter 20 has a port 25 coupled to a SCSI bus 50 which in turn connects each to a plurality of devices 61–63. Devices 61–63 can be any of a variety of peripheral devices conforming to the SCSI standard such as hard files, floppy disc drives, printers, CD ROMS, etc. As will be more completely described below the SCSI adapter 20 includes a diagnostic subsystem which allows the adapter 20 to test a SCSI controller included within the adapter 20. FIG. 2 is a block diagram of the adapter 20 showing several of the components therein including a peripheral device controller 201 which can, for example, be a SCSI controller. The controller 201 is coupled through the port 25 to the SCSI bus 50. The controller also, is coupled over a bus 206 to a dedicated micro-processor 202 (such as the Intel 80188) and to ROM 203 and RAM 204. The micro-processor 202 is coupled to a Buffer & Data Flow Control element 205 and to a System Interface Control 206. The System Interface Control 206 is coupled via the port 23 to the bus 30. The adapter 20 also includes an intelligent buffer 207 coupled to the SCSI controller 201 through the Buffer & Data Flow Control element 205. FIG. 2 does not show the diagnostic sub-system which can be incorporated as is shown in FIG. 3. FIG. 3 shows the bus 206 coupling the micro-processor 202 to the SCSI controller 201 and the relationship of the micro-processor 202 to the ROM 203 and RAM 204. FIG. 3 shows in detail a plurality of input/output terminals of the SCSI controller 201. Input/output terminals include reset (RST), command/data (C/D), input/output (I/O), message (MSG), attention (ATN), select (SEL), busy (BSY), request (REQ), acknowledge (ACK), and parity terminals. Each of these terminals is connected via a dedicated conductor to a corresponding terminal in the port or connector 25 through which these terminals are connected to the SCSI bus 50. In addition, the SCSI controller 201 includes eight data input/output terminal each of which is connected to a different conductor and therethrough to a corresponding terminal in the connector or port 25.

FIG. 3 also shows the diagnostic gate array 220. The diagnostic gate array 220 includes an input from the bus 206, coupling signals generated from the dedicated micro-processor 202. The diagnostic gate array includes

an output terminal for each of eight data conductors, as well as for each of the following: ACK, REQ, BSY, SEL, PARITY, MSG, I/O, and C/D. Conductors from each of these output terminals are connected to the corresponding conductor coupled between a like input-/output terminal of the SCSI controller 201 and the port or connector 25. The diagnostic gate array 220 also includes a further input terminal RST which is connected to the conductor coupling the SCSI controller terminal RST to the port or connector 25.

FIG. 4 is a block diagram of one preferred embodiment of the diagnostic gate array 220 of FIG. 3. More particularly, as shown in FIG. 4 the bus 206 provides 12 input signals to the diagnostic gate array 220. These signals include eight bits of data D0–D7, two register select signals, CS1 and CS2, a Write control signal and a WRAP control signal. The thirteenth input to the diagnostic gate array 220 is provided at the RST terminal which is coupled to the RST terminal at port 25 and to the RST terminal of the SCI controller 201.

The gate array 220 includes two registers, REG1 and REG2. Each of the registers has eight data input terminals, each coupled to one of the input terminals D0–D7. Two additional inputs to the gate array 220 are the control signals CS1 and CS2. The presence of either CS1 or CS2 selects the associated element either REG1 or REG2 for writing the contents of D0–D7. The information is written in the joint presence of the appropriate select signal (CS1 or CS2) along with the Write signal, another input from the bus 206.

Each of the registers, REG1 and REG2, has each of its eight output terminals connected to a different one of a plurality of output gates OA1–OA16. The other input terminal to each of the gates OA1–OA16 is provided by the output of a gate G4 which is one element of a bistable element or flip-flop FF formed by gates G1–G4. The two inputs to the flip-flop FF are the RST signal and the WRAP signal. Assuming that RST is absent (or inactive) then assertion of WRAP partially enables each of the gates OA1–OA16 to repeat, at its output, the other input provided from the associated terminal of one of the registers REG1–REG2. Eight of the outputs of the gate array 220 are the signals DATA0–DATA7 which, as shown in FIG. 3 are dot OR'ed with the signals provided by the DATA terminals of the SCI controller 201. The other eight outputs of the gate array 220 are the control signals ACK, REQ, BSY, SEL, PARITY, MSG, I/O and C/D.

From the foregoing it should be apparent that the micro-processor 202 can, by providing the appropriate signals over the bus 206, drive any of the 16 output terminals of the gate array 220 in any desired sequence or pattern. More particularly, the data stored in any stage of either of the registers REG1 or REG2 is controlled by transmitting the appropriate bit on that one of the conductors, D0–D7 associated with the selected stage of the register, and asserting either CS1 or CS2 depending on which of the registers the stage appears in, along with the Write signal. Once the registers REG1 and REG2 have been loaded with the appropriate data, assertion of WRAP will produce the corresponding data at the output of the gate array 220.

The RST input, since it is connected via the port 25 to the RST conductor on the SCSI bus, will follow the state of the RST conductor on the SCSI bus.

Accordingly, even if the gate array 220 is being driven by the micro-processor 202, the presence of the reset signal on the RST bus will be detected by the gate array 220, that detection will cause the state of the flip - flop FF to change. The change in state of the flip-flop FF will disable each of the gates OA1–OA16. Accordingly, the gate array 220 will respond to the reset signal on the SCSI bus as any other SCSI device is required to respond by the standard.

In order to test the SCSI controller 201 the micro-processor 202 (driven by a micro-code) from either ROM203 or RAM204, first commands the SCSI controller 201 to arbitrate for control of the SCSI bus. This ensures that testing of the SCSI controller 201 will not interfere with any other transactions which may be simultaneously taking place across the SCSI bus. On successfully arbitrating for control of the bus, the SCSI controller 201 then proceeds to select itself, e.g. it identifies the target device as the device with its own ID. The micro-processor 202 then drives the gate array 220 with a sequence of signals. Those signals can simulate the normal response of any target device to the SCSI controller 201. The micro-processor 202 then monitors the status of the SCSI controller 201 to ensure that the SCSI controller 201 properly responds to the simulated response generated by the gate array 220. In addition, or in lieu of the foregoing the micro-processor 202 can drive the gate array 220 with a simulation of an error condition (PARITY overrun, etc.) and also monitor the response of the SCSI controller 201. In this fashion the SCSI controller may be tested whether or not any device is connected to the SCSI bus (so long as the bus is properly terminated) and if a device is so connected regardless of its identity or characteristics. Furthermore, by controlling the pattern of signals written to the gate array 220 the response of the SCSI controller 201 to fault-free or faultv devices can be detected by the micro-processor 202.

Although not illustrated, in the second form of the invention, the gates and registers of FIG. 4 are integrated into the structure of the SCSI controller 201. As a result the dot OR'ed connections of FIG. 3 are no longer external to the SCSI controller but rather are, in the second form of the invention, internal to the SCSI controller 201. In other respects the two forms of the invention have similar operational characteristics.

It should be therefore apparent that many other and further changes can be made within the spirit and scope of the present invention. The present invention is not to be limited by the examples described herein but is to be construed in accordance with the claims attached hereto.

We claim:

1. A device adapter capable of controlling a variety of peripheral devices each designed to comply with a specific standard, said device adapter including:

a connector for connection to a multiconductor bus;

a micro-processor and dedicated memory coupled thereto;

a device controller coupled to and driven by said micro-processor, said device controller including dedicated input/output terminals for driving selected conductors of said multiconductor bus and for responding to selected conductors of said multiconductor bus, and

a gate array with a set of inputs coupled to said micro-processor and a set of outputs coupled in common with said input/output terminals of said device controller.

2. A device adapter as recited in claim 1 wherein said specific standard is a Small Computer System Interface

standard and wherein said gate array includes a Reset input coupled to said multiconductor cable.

3. A device adapter capable of controlling a variety of peripheral devices each designed to comply with a Small Computer System Interface (SCSI) standard, said device adapter including:

a connector for connection to a SCSI bus;

a micro-processor and dedicated memory coupled thereto;

a SCSI device controller coupled to and driven by said micro-processor, said SCSI device controller including dedicated input/output terminals for driving selected conductors of said SCSI bus and for responding to selected conductors of said SCSI bus, and

a gate array with a set of inputs coupled to said micro-processor and a set of outputs coupled in common with said input/output terminals of said SCSI device controller.

4. A diagnostic subsystem for use with a Small Computer System Interface (SCSI) adapter where said Small Computer System Interface (SCSI) adapter includes a micro-processor and dedicated memory coupled thereto, and a SCSI controller coupled to said micro-processor and to a SCSI bus, said diagnostic subsystem including:

a gate array coupled, to said micro-processor and to said SCSI controller and, also coupled to said SCSI bus,

said subsystem further including:

means responsive to a diagnostic command received at said SCSI controller for selecting said SCSI controller and for generating and coupling, through said gate array, signals simulating a response of an attached device.

5. A diagnostic subsystem as recited in claim 4 wherein said means responsive to a diagnostic command for generating signals simulating an attached device includes means for simulating a fault free attached device.

6. A diagnostic subsystem as recited in claim 4 wherein said means responsive to a diagnostic command for generating signals simulating an attached device includes means for simulating a faulty attached device.

7. A diagnostic subsystem as recited in claim 4 wherein said gate array includes a plurality of register stages, each coupled to an input terminal of said gate array, first logic means responsive to command signals for enabling at least one of said register stages to store information representative of a signal appearing at said input terminal, and second logic means for coupling information stored in said at least one register stage to an output terminal.

8. A diagnostic subsystem as recited in claim 7 wherein said second logic means includes:

a reset input terminal coupled to a reset signal carrying conductor included in said SCSI bus, and means responsive to an active reset signal for disabling said second logic means from coupling information to said output terminal.

9. A diagnostic subsystem as recited in claim 8 wherein said second logic means includes a wrap test input terminal, and means responsive to a transition of said wrap test signal for enabling said second logic means.

* * * * *

UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF CORRECTION

PATENT NO. : 5,033,049

DATED : July 16, 1991

INVENTOR(S) : Keener et al

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

Col. 1, line 52, "at" should be -et-;

line 66, "peripherial" should be -peripheral-.

Col. 3, line 20, "an" should be deleted.

Col. 4, line 62, "terminal" should be -terminals-.

Signed and Sealed this

Twenty-fourth Day of November, 1992

Attest:

DOUGLAS B. COMER

Attesting Officer

Acting Commissioner of Patents and Trademarks

US005440697A

United States Patent [19]

Boegel et al.

[11] **Patent Number: 5,440,697**

[45] **Date of Patent: Aug. 8, 1995**

[54] **METHOD AND APPARATUS FOR SIMULATING I/O DEVICES**

[75] Inventors: **Mark A. Boegel; Douglas O. Bolstad; Stephen A. Knight; Harvey G. Kiel,** all of Rochester; **Robert R. Nelson,** Oronoco; **Pamela A. Wright,** Rochester, all of Minn.

[73] Assignee: **International Business Machines Corporation,** Armonk, N.Y.

[21] Appl. No.: **138,602**

[22] Filed: **Oct. 18, 1993**

Related U.S. Application Data

[63] Continuation of Ser. No. 781,460, Oct. 23, 1991, abandoned.

[51] **Int. Cl.**[6] **G06F 13/10**

[52] **U.S. Cl.** **395/500;** 395/828; 395/883; 395/183.09; 395/309

[58] **Field of Search** 395/325, 275, 500, 375, 395/650, 200, 575

[56] **References Cited**

U.S. PATENT DOCUMENTS

3,932,843	1/1976	Trelut et al.	395/500
4,031,517	6/1977	Hirtle	395/500
4,040,021	8/1977	Birchall et al.	395/550
4,484,266	11/1984	Becker et al.	395/500
4,814,983	3/1989	Catlin	395/500
4,885,681	12/1989	Umero et al.	395/700
4,901,260	2/1990	Lubachevsky et al.	364/578
4,956,787	9/1990	Ito et al.	364/474.24
4,985,860	1/1991	Vlach	364/578
5,088,033	2/1992	Binkley et al.	395/500
5,093,776	3/1992	Morss et al.	395/500
5,097,412	3/1992	Orino et al.	395/500
5,129,064	7/1992	Fogg, Jr. et al.	395/275

FOREIGN PATENT DOCUMENTS

2494869	11/1981	France	G06F 13/00
9212480	7/1992	WIPO	G06F 13/12

OTHER PUBLICATIONS

IBM Technical Disclosure Bulletin vol. 35, No. 2, Jun. 1992, Armonk, N.Y., US pp. 230-232 "Non-Invasive Automation of Softwre Test With Shared Peripherals".

Patent Abstracts of Japan vol. 14, No. 402 (P-1099) 30 Aug. 1990 & JP-A-02 155 050 (NEC Corp.) 14 Jun. 1990 *abstract*.

M. J. Corrigan, et al; IBM Technical Disclosure Bulletin vol. 33, No. 7, Dec. 1990, "Adopting Live Environment for Simulator Debug/Test".

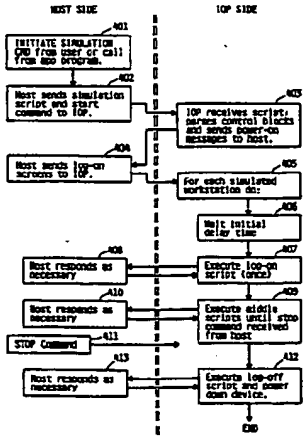
A. Poursepanj, IBM Technical Disclosure Bulletin vol. 33, No. 2, Jul. 1990, "Run-Time Software Architecture for a Workstation-Based Hardware Simulator".

Primary Examiner—Jack B. Harvey
Assistant Examiner—Glenn A. Auve
Attorney, Agent, or Firm—Roy W. Truelson

[57] **ABSTRACT**

A computer system comprises a CPU, a main memory, and plurality of I/O Processors (IOPs), coupled to each other by a system I/O bus. The IOPs perform slave processing functions relating to I/O devices. A simulation protocol is defined for the IOPs, whereby the host CPU can command an IOP to execute a simulation script. The simulation script defines one or more I/O devices to be simulated, and specifies a simulated workload associated with the devices. The IOP executes the simulation script by sending simulated input streams to the host and receiving output destined for the simulated I/O devices from the host on the system I/O bus. An IOP may simulate multiple I/O devices, and may simulate I/O devices concurrently with servicing real I/O devices. In typical use, one or more applications programs will execute on the CPU concurrently with the execution of one or more simulation scripts in the IOPs attacked to the system I/O bus. The behavior of the system under such conditions can be used for capacity planning forecasts, for developing and debugging the application software, for reproducing and diagnosing intermittent error conditions, or for performing other tasks in which it is necessary to determine the characteristics of the system under hypothetical conditions.

22 Claims, 8 Drawing Sheets



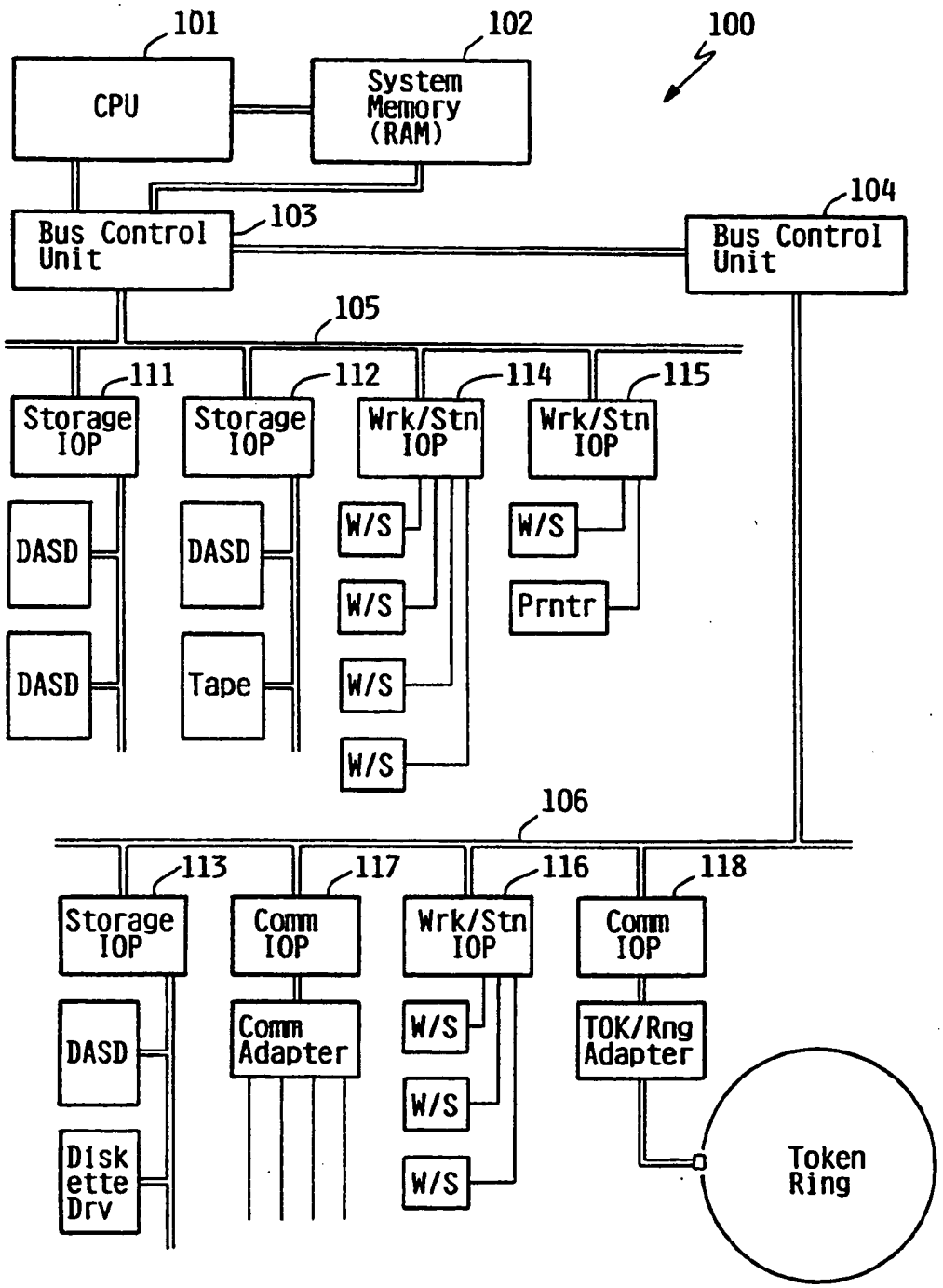

101
102
100
CPU
System Memory (RAM)
104
Bus Control Unit
103
Bus Control Unit
105
111
112
114
115
Storage IOP
Storage IOP
Wrk/Stn IOP
Wrk/Stn IOP
DASD
DASD
DASD
Tape
W/S
W/S
W/S
W/S
W/S
Prntr
106
113
117
116
118
Storage IOP
Comm IOP
Wrk/Stn IOP
Comm IOP
DASD
Diskette Drv
Comm Adapter
W/S
W/S
W/S
TOK/Rng Adapter
Token Ring

FIG. 1

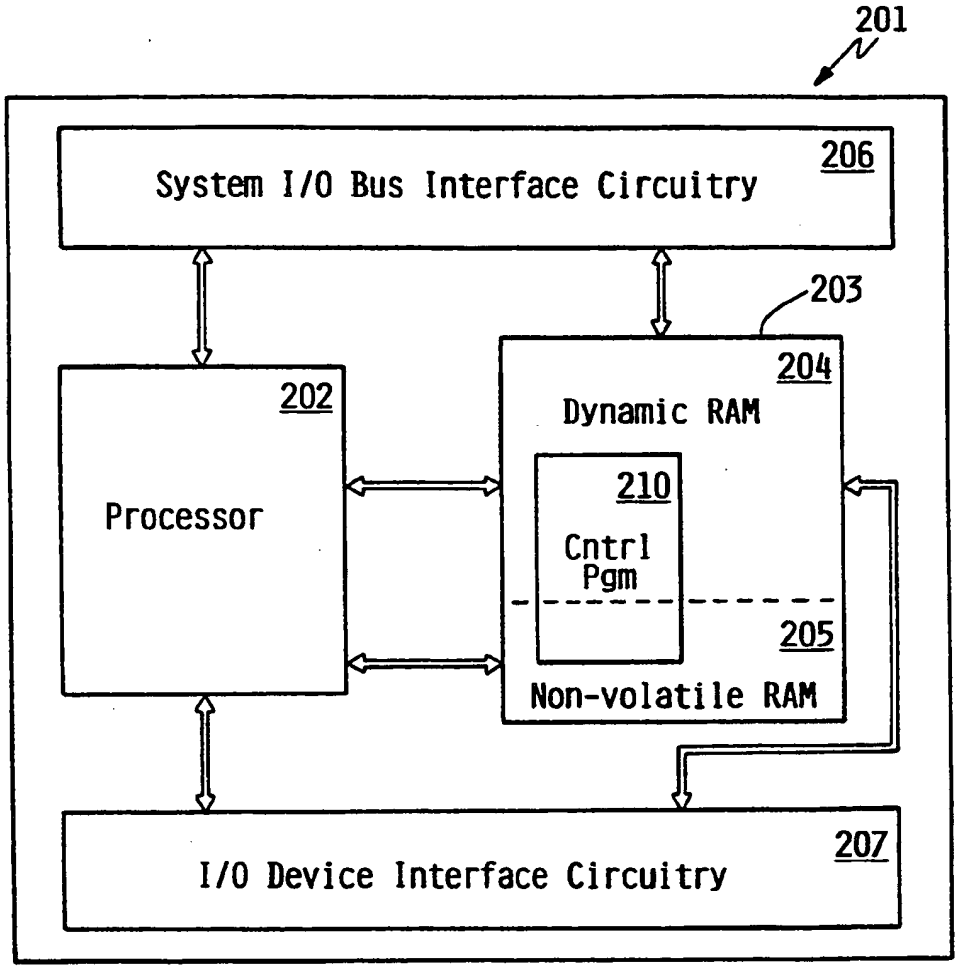

201
206
System I/O Bus Interface Circuitry
203
204
Dynamic RAM
202
Processor
210
Cntrl
Pgm
205
Non-volatile RAM
207
I/O Device Interface Circuitry

FIG. 2

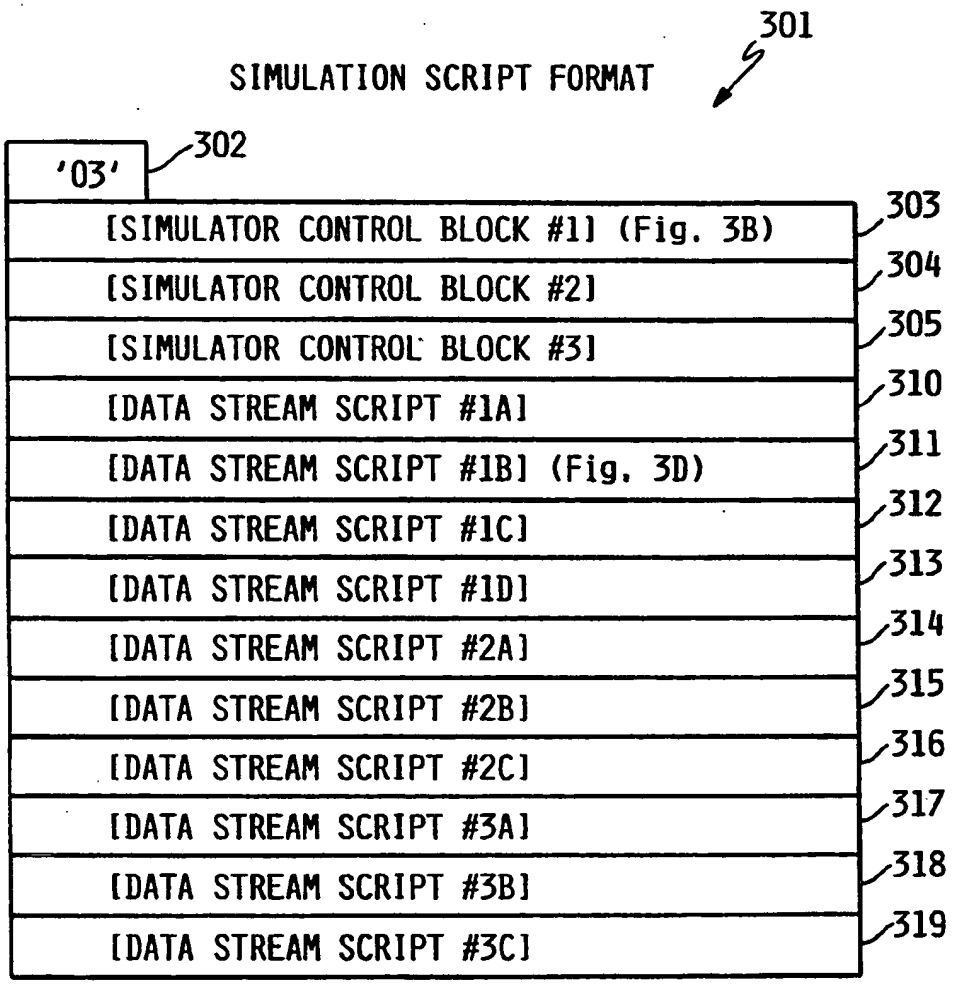

301
SIMULATION SCRIPT FORMAT
'03'
302
[SIMULATOR CONTROL BLOCK #1] (Fig. 3B)
303
[SIMULATOR CONTROL BLOCK #2]
304
[SIMULATOR CONTROL BLOCK #3]
305
[DATA STREAM SCRIPT #1A]
310
[DATA STREAM SCRIPT #1B] (Fig. 3D)
311
[DATA STREAM SCRIPT #1C]
312
[DATA STREAM SCRIPT #1D]
313
[DATA STREAM SCRIPT #2A]
314
[DATA STREAM SCRIPT #2B]
315
[DATA STREAM SCRIPT #2C]
316
[DATA STREAM SCRIPT #3A]
317
[DATA STREAM SCRIPT #3B]
318
[DATA STREAM SCRIPT #3C]
319

FIG. 3A

SIMULATOR CONTROL BLOCK FORMAT

303

Field Name	Byte location	Data	
Block Length	0 - 1	X '0036'	321
Device ID	2	X 'C2'	322
Model ID	3	X '80'	323
Keyboard ID	4	X '02'	324
Ext Keybd ID	5	X '00'	325
Device Addr	6	X '18'	326
Delay Time	7 - 8	X '012C'	327
Stream Count	9	X '04'	328
Data Stream Control Blocks (3 or more) (Fig. 3C)	10 - (9+11*N)	[BLOCK A]	329
		[BLOCK B]	330
		[BLOCK C]	331
		[BLOCK D]	332
End Byte	10+11*N	X 'FF'	333

FIG. 3B

DATA STREAM CONTROL BLOCK FORMAT

330

Field Name	Byte location	Data	
Script Length	0 - 1	X '02D6'	341
Offset	2 - 5	X '0000018D'	342
Keying Rate	6	X '02'	343
Time Multiplier	7	X '04'	344
Loop Count	8 - 9	X '000A'	345
End Byte	10	X 'FE'	346

FIG. 3C

311

DATA STREAM SCRIPT FORMAT

Field Name	Byte location	Data	
Start Command	0 - 15	[STRT CMD]	351
Data Stream	15 - M	[STREAM OF 1-BYTE KEYSTROKES AND CMDS]	352
End Command	M+1 - M+5	[END CMD]	353

FIG. 3D

351

DATA STREAM COMMAND FORMAT

Field Name	Byte location	Data	
Escape	0	X '00'	361
Command Length	1	X '10'	362
Command Code	2	X '00'	363
Opt Cmd Parms	3 - (K+2)	[VARIABLE]	364
Opt Cmd Code	K+3	X '00'	365
Command Length	K+4	X '10'	366
Escape	K+5	X '00'	367

FIG. 3E

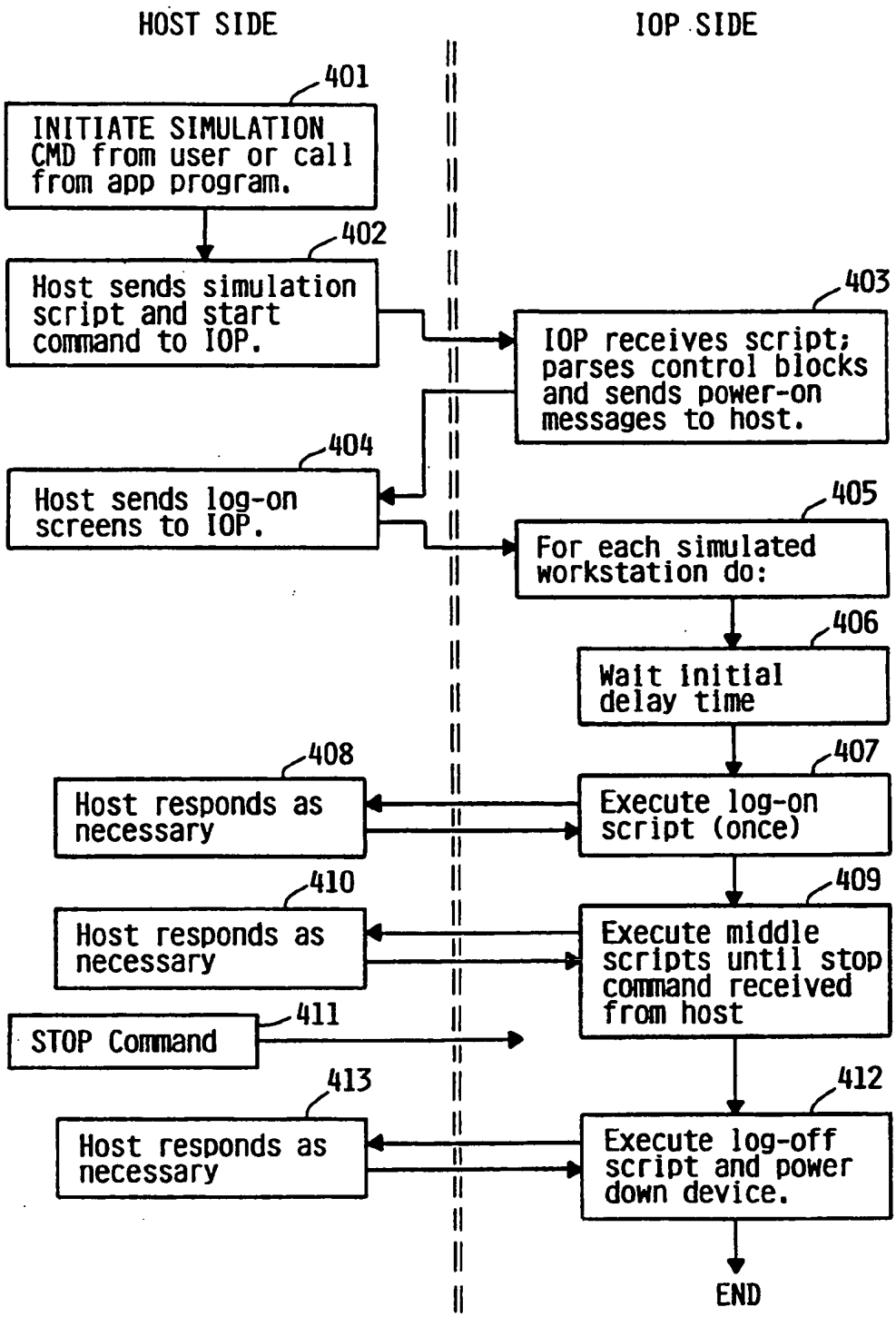

HOST SIDE
IOP SIDE
401
INITIATE SIMULATION CMD from user or call from app program.
402
Host sends simulation script and start command to IOP.
403
IOP receives script; parses control blocks and sends power-on messages to host.
404
Host sends log-on screens to IOP.
405
For each simulated workstation do:
406
Wait initial delay time
407
Execute log-on script (once)
408
Host responds as necessary
409
Execute middle scripts until stop command received from host
410
Host responds as necessary
411
STOP Command
412
Execute log-off script and power down device.
413
Host responds as necessary
END

FIG. 4

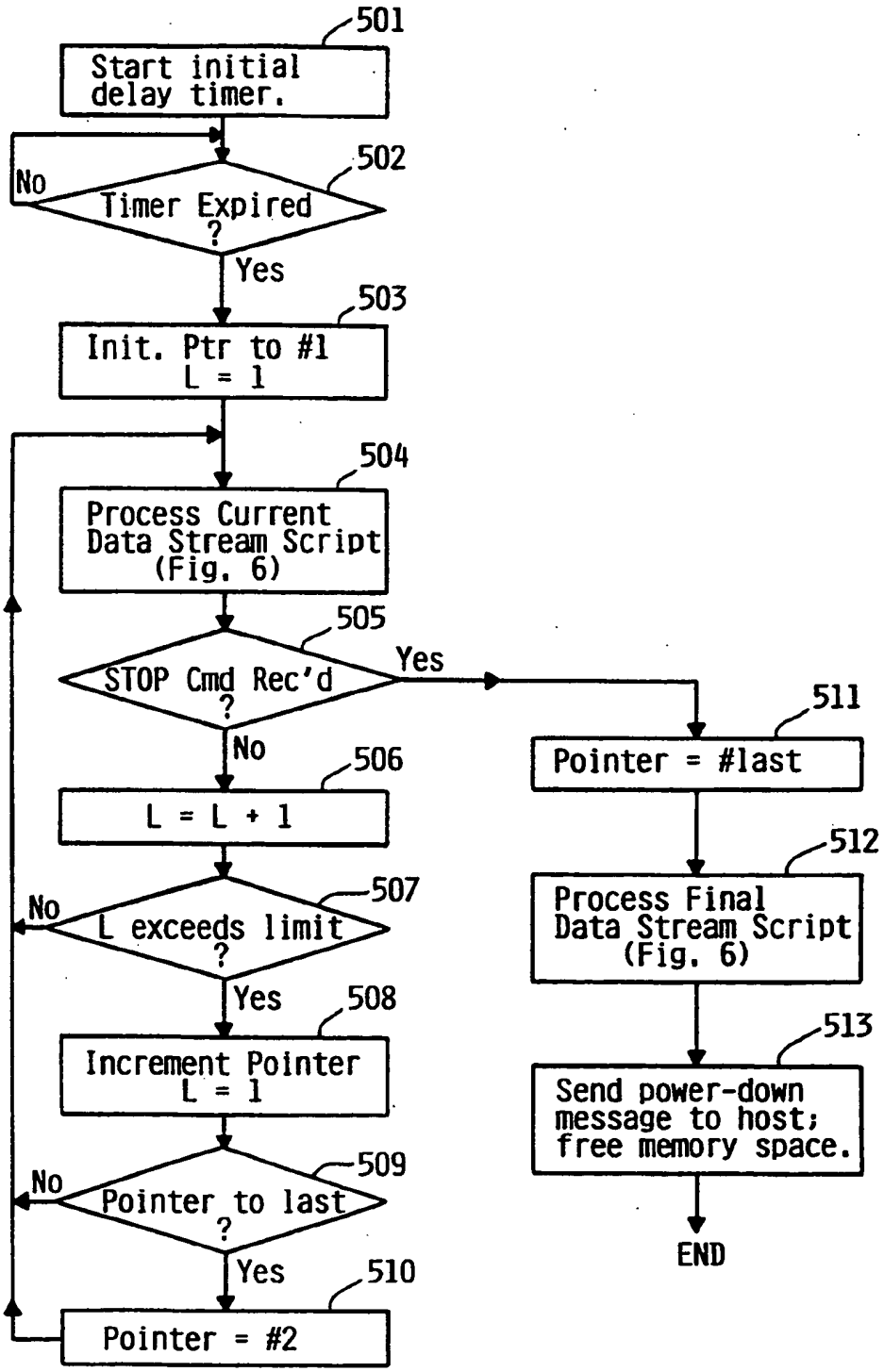

501
Start initial delay timer.
502
No
Timer Expired ?
Yes
503
Init. Ptr to #1
L = 1
504
Process Current Data Stream Script (Fig. 6)
505
STOP Cmd Rec'd ?
Yes
No
506
L = L + 1
507
No
L exceeds limit ?
Yes
508
Increment Pointer
L = 1
509
No
Pointer to last ?
Yes
510
Pointer = #2
511
Pointer = #last
512
Process Final Data Stream Script (Fig. 6)
513
Send power-down message to host; free memory space.
END

FIG. 5

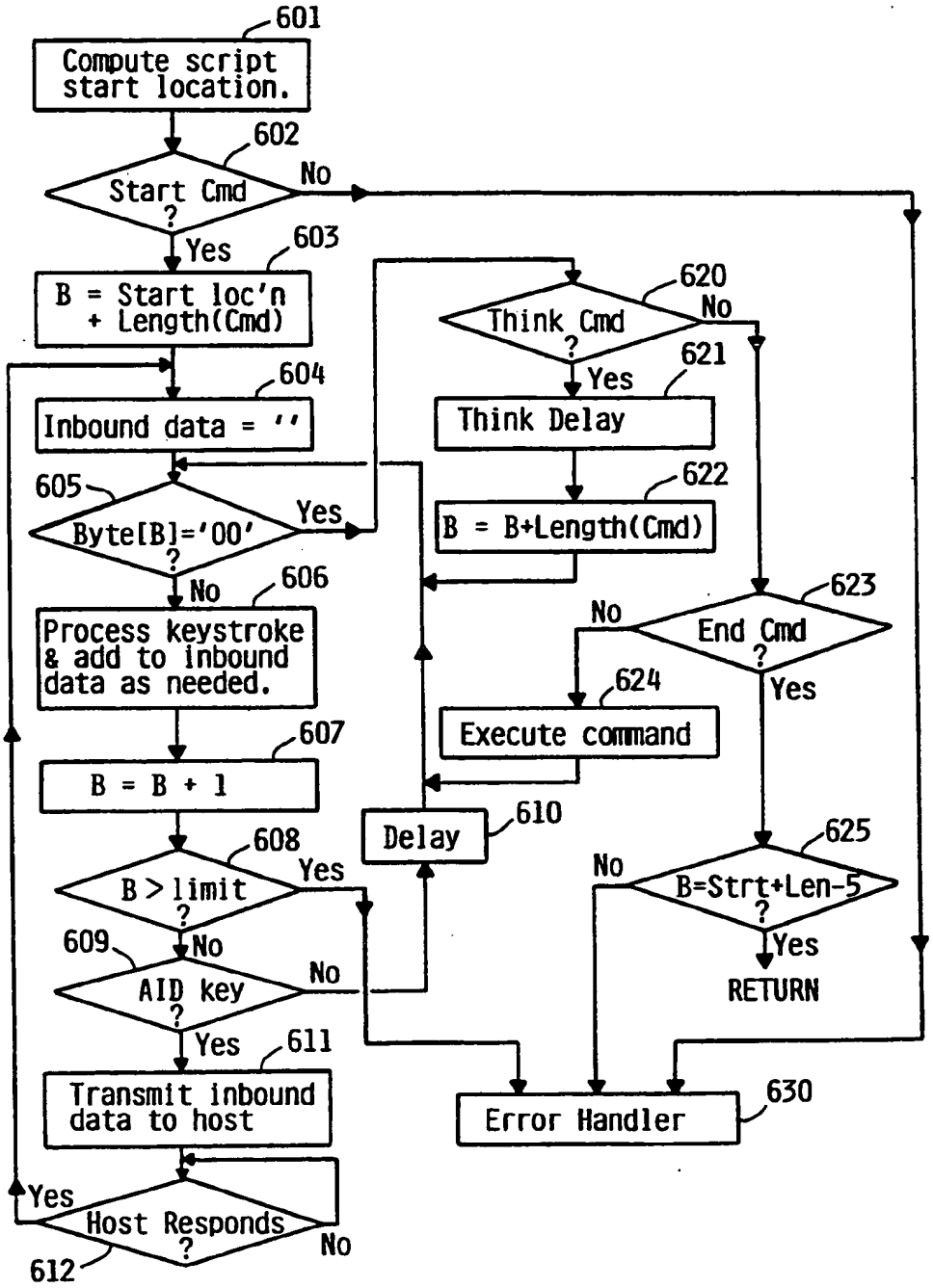

601
Compute script start location.
602
Start Cmd ?
No
Yes
603
B = Start loc'n + Length(Cmd)
604
Inbound data = ''
605
Byte[B]='00' ?
Yes
No
606
Process keystroke & add to inbound data as needed.
607
B = B + 1
608
B > limit ?
Yes
No
609
AID key ?
No
Yes
611
Transmit inbound data to host
Yes
Host Responds ?
No
612
620
Think Cmd ?
No
Yes
621
Think Delay
622
B = B+Length(Cmd)
623
End Cmd ?
No
Yes
624
Execute command
610
Delay
625
B=Strt+Len-5 ?
No
Yes
RETURN
630
Error Handler

FIG. 6

METHOD AND APPARATUS FOR SIMULATING I/O DEVICES

This application is a continuation of U.S. patent application Ser. No. 07/781,460 filed Oct. 23, 1991, now abandoned.

FIELD OF THE INVENTION

The present invention relates to data processing system design, and in particular to a design for supporting the simulation of physical devices attached to a computer system and workloads imposed on the system.

BACKGROUND OF THE INVENTION

Modern computer systems are increasingly being used to perform a large variety of difficult data gathering and analysis tasks. With the growing prevalence and complexity of computers, it is frequently desirable to obtain information and answer questions about the computer system itself. Due to the complexity of the machine, it is difficult to obtain useful information purely by manual methods. Since the computer has been used to solve such a large variety of problems, it is only natural that the computer be used for introspective tasks as well. For example, computer systems have been designed with capabilities to diagnose hardware failures, provide performance and usage information, assist debugging of programs, etc.

When solving introspective problems, a computer is often required to answer hypothetical questions. It may need to know how it would respond to a given external workload, how long it would take to run a particular program, what would be the effect of temporary loss of a hardware component, etc. One type of application in which such questions must be answered is the capacity planning application. Capacity planning involves forecasting future workload to be imposed on a computer system. predicting how the system will respond to such workload, and determining how such response may be improved (as by attaching additional hardware to the system). Another such application is hardware and software development, in which it is desirable to know how a system will respond to a hardware or software component under development in a variety of prospective environments. Still another possible application is in defect diagnosis, particularly defect diagnosis of an intermittent error condition, in which re-creation of the intermittent error condition requires that a particular environment be assumed. Another possible application is performance analysis and diagnosis, in which it is necessary to determine how the system behaves under given conditions and how system behavior can be altered.

In theory, it is possible to duplicate the hypothetical conditions with actual hardware running appropriate software on behalf of real users. In reality, such a solution is seldom practicable. For example, a commercial enterprise planning for future expansion may wish to know how its system would respond to increased workload from a larger number of attached terminals and interactive users. In theory it could buy the required equipment, hire the terminal operators, create additional workload for input to the system, and observe the results. In reality, such an approach is prohibitively expensive. In another example, a software development house may wish to know how its software will behave on a particular family of computer systems, under a variety of system configurations and workloads. It could buy all the hardware required to duplicate the various configurations, hire the required number of interactive users, and observe the results; again, this approach would be prohibitively expensive and time consuming. In addition to being prohibitively expensive and time consuming, the duplication of real conditions as described above is frequently undesirable because it is impossible to obtain repeatable results. When real users enter data at a keyboard, there will naturally be some variation in the actual timing of input streams, causing observed results to vary. In some applications, such variation is a serious drawback. For example, a developer who is attempting to diagnose an intermittent error condition will want to establish a repeatable sequence of events by which the error condition can be induced.

Another approach to providing data for hypothetical problems is the use of special purpose hardware simulators, which take the place of attached devices with real users. Such hardware simulators obviate the need for live interactive users, and can be made to generate simulated workload in a repeatable manner. However, hardware simulators are still a very expensive method of providing this information. Typically hardware simulators are more expensive than the devices they replace. As a result, the use of hardware simulators is generally confined to system development laboratories.

It is possible to create simulation software. This software attempts to duplicate the hypothetical conditions by issuing instructions from the central processing unit (CPU), causing simulated workload to be processed by the system. Such software is adequate for some tasks, but often lacks the ability to simulate many conditions on the system. The computer system typically comprises a single CPU, which is coupled to various memory devices and slave processors via one or more communications buses. The slave processors are in turn coupled to I/O devices such as disk drives, interactive workstations, printers, etc., or may be coupled to other slave processors. System behavior will depend not only on the number and type of such devices, but the connection topology as well. Simulation software running in the CPU will not always accurately represent the state of other parts of the system, in particular the slave processors and I/O devices.

When a computer performs introspective tasks, it may obtain some information from the operator, but generally the source of its information is the computer itself. As with any task performed on a computer, the quality of the output information can be no better than the quality of the input. Unfortunately, the computer systems of the prior art lack the ability to obtain accurate information for answering many hypothetical questions at reasonable cost.

SUMMARY OF THE INVENTION

It is therefore an object of the present invention to provide an enhanced method and apparatus for performing introspective tasks on a computer system.

Another object of this invention is to provide an enhanced method and apparatus for simulating devices attached to a computer system.

Another object of this invention is to provide an enhanced method and apparatus for simulating a workload input to a computer system.

Another object of this invention is to increase the accuracy of information in a computer system concerning hypothetical devices and workloads.

Another object of this invention is to reduce the cost of providing accurate information concerning hypothetical devices and workloads in a computer system.

Another object of this invention is to provide an enhanced method an apparatus for performing capacity planning tasks on a computer system.

Another object of this invention is to increase the accuracy of capacity planning forecasts for a computer system.

Another object of this invention to provide an enhanced method and apparatus for developing software for a computer system.

Another object of this invention to provide an enhanced method and apparatus for diagnosing problems in a computer system.

Another object of this invention is to perform computer system device and workload simulations with increased accuracy and repeatability.

A computer system comprises a CPU, a main memory, and a (IOPs), coupled to each other by one or more system I/O busses. The IOPs perform slave processing functions relating to the I/O devices. Each IOP contains a programmable processor. Each I/O device attaches to the system via an IOP, and all communications between it and other elements of the system must pass through the IOP.

A simulation protocol is defined for the IOPs, whereby the CPU executing instructions resident in main memory can command an IOP to execute a simulation script. The simulation script defines one or more I/O devices to be simulated, and specifies a simulated workload associated with the devices. The IOP executes the simulation script by interacting with the system I/O bus as if real I/O devices attached to the IOP were performing the work indicated in the script. For example, if an interactive workstation is being simulated, the IOP may send simulated input data strings to the CPU at intervals, and receive output from the CPU intended for interactive display on the simulated workstation.

It is not necessary that any of the simulated I/O devices actually exist on the system. A single IOP may simulate more than one I/O device, and more than one type of I/O device, and multiple IOPs may simulate multiple I/O devices simultaneously. Real I/O devices may operate concurrently with the simulation, form the same IOP and from different IOPs. Thus, virtually any hypothetical configuration of I/O devices to the system may be simulated.

In typical use, one or more applications programs will execute on the central processing unit concurrently with the execution of one or more simulation scripts in the IOPs attached to the system I/O bus. The IOPs will simulate, through execution of the simulation scripts, input to the applications programs that would be expected from real I/O devices during normal use. The IOPs will also receive and acknowledge output produced by the application programs, destined for the simulated I/O devices. The behavior of the system under such conditions can be used for capacity planning forecasts, for developing and debugging the application software, for reproducing and diagnosing intermittent error conditions, or for performing other tasks in which it is necessary to determine the characteristics of the system under hypothetical conditions.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** shows the major hardware components of a computer system having means for simulating I/O devices according to the preferred embodiment of this invention;

FIG. **2** shows the major components of a typical I/O Processor assembly of a computer system according to the preferred embodiment;

FIG. **3A** shows the high-level format of a typical simulation script according to the preferred embodiment;

FIG. **3B** shows the format of a typical simulator control block within the simulation script according to the preferred embodiment;

FIG. **3C** shows the format of a typical data stream control block within the simulator control block according to the preferred embodiment;

FIG. **3D** shows the high-level format of a typical data stream script according to the preferred embodiment;

FIG. **3E** shows the format of a data stream command contained in the data stream script according to the preferred embodiment;

FIG. **4** is a high-level flowchart of the steps involved in simulating I/O devices according to the preferred embodiment;

FIG. **5** is a flowchart of the steps performed by the IOP control program when simulating an I/O device according to the preferred embodiment;

FIG. **6** shows the steps performed by the IOP control program when parsing a data stream script according to the preferred embodiment.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

A diagram of the major hardware components of a computer system with the capability to simulate I/O devices according to the preferred embodiment of the present invention is shown in FIG. **1**. Computer system **100** comprises a central processing unit (CPU) **101** coupled to a system random access memory **102**. System memory **102** is in the address space of CPU **101**. Although it is shown as a single unit, it should be understood that it may in fact comprise a hierarchy of memory devices, such as a small, relatively fast cache memory and a slower but larger main memory, as is known in the art. CPU **101** and memory **102** are coupled to bus interface unit **103**, which arbitrates control of system I/O bus **105** and handles communications between bus **105** and CPU **101** or memory **102**. System **100** may comprise more than one system I/O bus. It is shown in FIG. **1** with a second bus interface unit **104** connected to unit **103**. The second unit **104** arbitrates control of a second system I/O bus **106**, and communicates with the first system I/O bus **105**, CPU **101** and memory **102** through bus interface unit **103**. Buses **105,106** provide a communications path to a plurality of I/O processors **111–118**. I/O processors handle communications with I/O devices, and may control some of the function of such devices. Various different types of I/O processors exist on system **100**. Each storage I/O processor **111–113** controls a high-speed I/O channel for servicing storage devices such as magnetic disk drive direct access storage devices (DASD), magnetic tape drive, and diskette drive. Workstation I/O processors **114–116** service local interactive workstations and printer through a plurality of direct local connections. Communications I/O processors **117–118** service remote com-

munications lines and token ring local area network, which may attach to additional I/O devices or other computer systems (which will appear to system **100** as I/O devices). It should be understood that the number, type and configuration of I/O devices, I/O processors and other components may vary. A computer system according to this invention may contain a single system I/O bus or may contain multiple buses. Different types of I/O processors may be present in the system, including multi-function I/O processors which perform the function of more than one type of I/O processor. Different types of I/O devices may be present, such as optical storage devices, optical character readers, voice I/O devices, etc. In addition, system **100** may be a multiprocessor system, comprising a plurality of central processing units. In the preferred embodiment, system **100** is an IBM Application System/400 system.

The major components of a typical I/O Processor assembly (IOP) **201** of the preferred embodiment are shown in FIG. **2**. The I/O Processor assembly comprises a programmable processor **202**, a locally addressable memory **203**, system I/O bus interface circuitry **206**, and device interface circuitry **207**, which are coupled to each other via various communications paths as shown. Locally addressable memory **203** comprises non-volatile portion **205** for storing instructions and vital data required for start-up of the IOP, and dynamic portion **204**. The purpose of the IOP is to relieve the CPU of the burden of performing all I/O support by itself. The IOP supports I/O devices by driving high-speed I/O channels, workstation lines, or other communication lines. The IOP may also handle direct memory accesses between the system memory and an I/O device. In the case of workstations, the IOP may also perform various tasks relating to updating the display, such as processing of data keystrokes, cursor movement, scrolling, etc. Thus, the CPU is shielded from the details of such I/O support, and will only hear from an I/O device when the device has some input ready for the system.

In operation, the IOP's programmable processor **202** executes instructions contained in a control program **210** stored in locally addressable memory **203**. Memory **203** may also serve as a cache for temporary storage of data being transferred between the system I/O bus and an I/O device. Processor **202** is capable of moving data or issuing commands, status information, etc. to either system I/O bus interface circuitry **206** or device interface circuitry **207**. Data may also be moved through memory **203**, by-passing processor **202**.

In the preferred embodiment, IOP **201** is a workstation controller to which a plurality of interactive workstations may be attached. The IOP maintains a screen buffer and field format table for each workstation it serves in dynamic memory **204**. The screen buffer stores the contents of the current display screen at the workstation; the field format table identifies the location of entry fields. The buffer and field format table enable the workstation controller IOP to process certain keystrokes received from the workstation without intervention from the host. For example, cursor movement keystrokes can be processed by updating the screen buffer.

It should be understood that, as used herein, an I/O Processor refers to an entity performing slave processing functions relating to I/O. Many different IOP designs exists in computer systems. In the preferred embodiment, the I/O Processor assembly comprises a single electronic circuit card on which are mounted a plurality of electronic components, said components being electrically connected with each other via various circuits contained in one or more planes of printed circuitry on the circuit card. However, the I/O Processor assembly could be contained in multiple circuit cards, or on a part of a single circuit card, or on a single chip.

From the perspective of that part of the system on the host side of the IOP (i.e., the CPU, system memory, system I/O bus, other IOPs, etc.), the IOP is the I/O device or devices which are attached to it. The CPU and/or main memory receive input data streams from the IOP over the system I/O bus, and send output data streams to the IOP over the system I/O bus. The essence of the invention disclosed herein is that the IOP, by generating and accepting data streams that would normally be received from and sent to an I/O device, can effectively simulate the I/O device to the host part of the system.

According to the preferred embodiment a simulation protocol is defined for the IOPs. This protocol defines a procedure whereby the CPU, executing instructions resident in main memory, can issue a command to an IOP to execute a simulation script. The simulation script, whose format is defined by the protocol, is transferred from the main memory to the IOP for execution. The script is a shorthand description of actions to be taken by the IOP to simulate one or more I/O devices. The script may be viewed as a series of commands which the IOP executes, but each "command" may call for a complex series of actions by the IOP rather than a single instruction executed by the IOP's programmable processor. A portion of the IOP's control program resident in IOP local memory **203** decodes each block of the script and may expand it into the series of actions required of the IOP to simulate the I/O device(s). The IOP then executes the actions called for by the expanded script. Execution involves interacting with the system I/O bus as if real I/O devices attached to the IOP were performing the work meant to be simulated by the script. Data streams typical of what the I/O device would generate are generated by the IOP instead and placed on the system I/O bus, bound for the CPU, main memory, or other system component. Outbound data streams from the host, destined for the I/O device, are received and acknowledged by the IOP and processed by it as if a real I/O device were attached, but no data is in fact transmitted by the IOP to an I/O device.

FIG. **3A** shows the high-level format of a typical simulation script **301** according to the preferred embodiment. In this embodiment, the IOP is a workstation controller to which a plurality of interactive workstations may be attached, it being understood that this invention could be practiced with other types of IOPs, including multi-function IOPs. The simulation script comprises a one-byte control block count field **302**, one or more simulator control blocks **303–305**, and one or more data stream scripts **310–319**. Block count **302** indicates the number of simulator control blocks **303–305** in script **301**. In the example of FIG. **3**, three simulator control blocks **303–305** are contained in the script, and block count **302** is accordingly set to "3". Each simulator control block defines a single I/O device to be simulated. Each data stream script defines a simulated input data stream (in this case, simulated keystrokes) that the IOP will process as if received from the keyboard of an interactive workstation to generate an inbound data stream. The inbound data stream is then

transmitted on a system I/O bus as if it originated from the interactive workstation. Multiple data stream scripts may be associated with each simulated I/O device. In the example of FIG. 3, four data stream scripts are associated with the first simulated I/O device, and three data stream scripts are associated with each of the other simulated I/O devices. In addition, control blocks associated with different simulated I/O devices may share one or more data stream scripts.

FIG. 3B shows the format of a typical simulator control block **303**, which is part of simulation script **301**. Simulator control block **303** comprises the fields described below. Block length field **321** (at byte locations **0-1**) is a two-byte field containing the total length (in bytes) of the simulator control block. Device Id field **322** and model ID field **323** identify the type and model of the I/O device (in this case, a workstation) being simulated. Keyboard ID field **324** and extended keyboard ID field **325** identify the type of keyboard attached to the simulated workstation. Device address field **326** identifies the logical address within the IOP of the simulated workstation. The system-wide address of a workstation is a combination of the address of the IOP and the address within the IOP of the workstation. Delay time field **327** contains the delay time (in seconds) from the start of a simulation until the IOP is to begin executing the data stream scripts called out by the simulation control block. Stream count field **328** identifies the number of data stream control blocks contained in simulator control block **303**. A variable number of data stream control blocks **329-332** (not fewer than three) follow stream count field **328**. The data stream control blocks occupy byte locations **10** through (**9+11*N**), where N is the number of such control blocks, as shown. Each data stream control block **329-332** identifies a data stream script to be executed by the IOP. End byte **333** signals the end of simulator control block **303**.

FIG. 3C shows the format of a typical data stream control block **330**, which is part of simulator control block **303**. The data stream control block is used to identify the data stream script and control its execution. While each data stream control block is associated with a data stream script, it is possible for one data stream script to be shared by multiple control blocks, enabling multiple simulated I/O devices to use the same data stream script. In the example of FIG. 3C, data stream control block **330** is associated with data stream script **311**. Data stream control block **330** comprises the fields described below. Script length field **341** identifies the length (in bytes) of the corresponding data stream script. Offset field **342** identifies the starting location of the data stream script in the simulation script record as an offset (in bytes) from the beginning of the record. Keying rate field **343** identifies the rate (in characters per second) at which keystrokes are to be simulated. Think time multiplier field **344** contains a multiplier used to calculate response times from the workstation (simulating thinking by the interactive user). Loop count field **345** indicates the number of times that the data stream script should be executed before continuing to the next script. End byte **346** signals the end of the data stream control block.

FIG. 3D shows the high-level format of a typical data stream script **311**. The data stream script is a variable length stream of commands and individual bytes representing keystrokes. It is delimited by start command **351** and end command **353**. Between the start command and end command may exist additional commands, but primarily the data stream portion **353** between the commands consists of individual bytes representing keystrokes from the simulated workstation. In the preferred embodiment, a think time command is defined for the data stream portion, which causes the IOP to insert a specified time delay in the data stream. The time delay specified in the command is multiplied by the think time multiplier specified in field **344** of the data stream control block.

FIG. 3E shows the format of a data stream command **351** contained in the data stream. The command comprises a 1-byte escape code to signal the start of the command **361**, a length field **362** indicating the length (in bytes) of the command, a command code **363** identifying the command, a second length field **366** and 1-byte escape field **367**. Fields **366** and **367** make the command a symmetric series of bytes, enabling it to be read either forward or backward. Command **351** optionally contains additional command parameters **364**. Where additional parameters **364** are present, a second command code field **365** is necessary to maintain symmetry. In the example of FIG. 3E, the command is a start command, having 10 bytes of additional parameters specifying the data stream script length and identifier.

The operation of the present invention according to the preferred embodiment will now be described. FIG. **4** is a high-level flowchart of the steps involved. A simulation is initiated from the host side of the system. Simulation may be initiated either by a call from an application program running on the host CPU or by a command entered from an interactive terminal (step **401**). The simulation script **301** may be imbedded in the application program or may be a separate file. The application program or user who initiated a simulation may initiate simulations in multiple IOPs simultaneously, which may use the same or different simulation scripts. FIG. **4** shows the simulation steps only for a single IOP, it being understood that multiple IOPs could be performing the steps shown in FIG. **4** simultaneously.

For example, a capacity planning application program may test a hypothetical configuration and workload by calling one or more simulation scripts, executing the scripts and measuring the system response. In another example, a software developer developing a software module may command the IOPs to execute one or more simulation scripts while the software module is executing; the simulation scripts could merely provide typical background work for the system or may invoke the software module under test directly. In a further example, a system developer might use the system console to invoke simulations for debugging intermittent error conditions; once a simulation script can be constructed to induce the intermittent error, it can repeatably be induced any number of times, and the behavior of the system observed.

Upon initiation of the simulation, the host sends a command to the IOP to begin simulation, along with the simulation script (step **402**). The script is stored somewhere on the host side of the system, which could be in main memory, disk storage or elsewhere, and is downloaded to the IOP via the system I/O bus. The IOP stores the script in its local memory **203** for the duration of the simulation. In the preferred embodiment, the simulation script is not kept in the IOP local memory **203** during normal operation, although that portion of the IOP's control program responsible for decoding the script and executing it is. The download-

ing of simulation scripts from the host permits greater flexibility in the content of the simulations, since it is possible for the host to store a variety of simulation scripts. However, in an alternative embodiment it would be possible to store a standard simulation script permanently in the IOP as part of its control program.

Upon receipt of simulation script **301**, the IOP stores the script in local memory **203** and parses the various control blocks of the script (step **403**). Script **301** will contain one or more simulator control blocks **303–305**, each block corresponding to one I/O device to be simulated. The IOP will send a power-up message to the host for each simulated workstation. The power-up message informs the host that the interactive workstation has just been powered-up, and is awaiting instructions from the host. At the same time, the IOP allocates a screen buffer and field format table for each simulated workstation in its local memory. The host will normally respond by sending a log-on screen to the workstation, just as it would if a real workstation had powered-up (step **404**).

The following described steps are executed separately for each simulated workstation. The IOP control program in effect has separate workstation sessions operating in parallel, just as it would for multiple real interactive workstations.

Upon receipt of each log-on screen, the IOP starts a timer for the simulated workstation to wait an initial delay time before executing a simulated log-on by an interactive user (step **406**). The delay time is the time specified in delay time field **327** of simulator control block **303**. Since a separate simulator control block **303** exists for each simulated device, it is possible to have different delay times, enabling staggered log-on of the simulated users.

After waiting the delay period, the control program in the IOP executes the first data stream script, which by convention is the log-on script (step **407**). The log-on script is executed only once. The IOP executes the data stream script by decoding any imbedded commands in the script and taking appropriate action, by processing the stream of data bytes contained in the script as if the same were received from a real workstation to generate inbound data, and transmitting on the system I/O bus the inbound data streams thus generated from the script. For example, in the case of the log-on data stream script, the script would typically contain a start command **351**, followed by a string of characters corresponding to the user's identifier, password, optional log-on parameters, and appropriate delimiters such as tab keys or enter keys (together constituting part **352**), followed by an end command **353**. Start and end commands **351, 353** are to be distinguished from commands received from the host; commands **351, 353** are used only as delimiters of the data stream script, not to command the IOP to take some particular action. The host responds to the log-on script in the normal manner (step **408**), as if a real user were logging on to the system.

After executing the first (log-on) data stream script, the IOP executes the "middle" scripts in sequence (step **409**). The middle scripts comprise all data stream scripts other than the first and last. As in the case of the log-on script, each middle script is executed by processing a stream of data bytes contained in the script, taking appropriate action for any imbedded commands, and transmitting to the host on the system I/O bus the outbound data streams generated as a result. For example, an imbedded command may require the IOP to pause a specified length of time in the middle of the data stream. The data stream script is repeated the number of times specified in loop count field **345** of data stream control block **330**. When the IOP finishes executing a data stream script, it proceeds to the next script. When all middle data stream scripts have been executed, the IOP loops back to the first middle script (the second data stream script) and repeats the sequence. The middle data stream scripts are repeated indefinitely until a stop command is received from the host.

As in the case of the log-on script, the host responds as necessary to the various input streams received as a result of the middle data stream scripts (step **410**). For example, a middle data stream script may produce inbound data streams to first call an editor program to edit a document, then make random character insertions and deletions in the document, and finally save the edited document. To each of these data streams, the host would respond, for example, by displaying the document initially, updating the document on the workstation display as alterations are made, and transmitting appropriate save messages to the workstation display. Note also that the data streams may cause various parts of the host to respond, as for example reading from and writing to disk storage as the document is initially called up and later saved.

When the IOP receives a command to stop simulation from the host (step **411**), it completes processing of the current data stream script, then executes the final data stream script, which is a log-off script (step **412**). After the host responds by acknowledging the log-off and returning the display to an idle state (step **413**), the IOP sends a power-down message to the host indicating that the simulated workstation has been powered down. This completes the simulation.

FIG. **5** shows in greater detail the steps performed by the IOP control program to simulate each workstation, which are represented in the high-level diagram of FIG. **4** as steps **405,406,407, 409,412**. When the IOP receives acknowledgment of power-on from the host and the log-on display, it sets a timer to the delay value specified in field **327** (step **501**). It then waits idle until the timer expires (step **502**). Upon expiration of the timer, a pointer to the current data stream script is initialized to the first script and a loop counter is initialized to 1 (**503**). The IOP then processes the current data stream script by parsing the script, building inbound data streams to the host, transmitting the streams, and receiving the host's response (step **504**). The details of processing a data stream script are shown in FIG. **6**. When the data stream script has been fully processed, the IOP checks for the presence of a stop command from the host (step **505**). If no stop command has been received, it then increments the loop count variable L (step **506**) and checks whether L exceeds the loop count specified in field **345** (step **507**); if not flow proceeds to step **504**, where the data stream script is repeated. If the loop count has been reached, the current data stream pointer is incremented to the next data stream script and the loop count is reset to 1 (step **508**). If the pointer now points to the last data stream script (step **509**), the pointer is reset back to the second script (step **510**). Flow then proceeds to step **504**. If at step **505**, it was determined that a stop command has been received, the current data stream script pointer is set to the last script (step **511**), which is the log-off script, and the script is processed in the same manner as the previous scripts (step **512**). The IOP then transmits a device power-

down message and frees that part of dynamic RAM allocated to the screen buffer and field format table for the simulated device (step 513). When all simulating devices have powered-down, that part of dynamic RAM allocated to storage of the simulation script is freed.

FIG. 6 shows the steps performed by the IOP control program when parsing a data stream script. The program determines the location of the data stream script in local memory from the location of the simulation script record 301 and offset field 342 of the data stream control block (step 601). It then parses the first 16 characters, which should be a start command (step 602); if the first 16 characters are not a start command, an error condition is signalled (step 630). A byte pointer for identifying bytes within the data stream script is set to the starting location of the data stream script plus the length of the start command (step 603); this is the location immediately after the start command. The inbound data stream destined for the system I/O bus is initialized to blank (step 604).

The program then parses the data stream script one byte at a time. If the byte being parsed is not '00'X (step 605), it is processed in the same manner as a keystroke received from the workstation as if the workstation really existed (step 606). The IOP of the preferred embodiment maintains state information concerning the workstation display in its screen buffer and field format table. Certain keystrokes, such as cursor movement or tab keys, cause the IOP to update this state information without sending anything to the host. Other keystrokes, such as alphanumeric characters, are typically added to an inbound data stream to be sent to the host. However, the data stream is not sent immediately to host, but is sent following an Attention Identifier (AID) key, such as the "Enter" key or a function key. Still other keystrokes may cause the IOP to edit the inbound data stream, as in the case of a backspace or cursor movement followed by typing over a previously entered input field.

When the keystroke has been processed, the byte pointer is incremented (step 607), and the IOP verifies that the byte pointer does not exceed the bounds of the data stream script record (step 608). If the length has been exceeded, an error is indicated (step 630). If not, and the key just processed was not an AID key (step 609), the simulation program waits a period of time equal to 1/(keying rate specified in field 343) at step 610. This delay simulates a typical delay between keystrokes from an interactive user. It then goes to step 605 to parse the next byte of the data stream script.

If an AID key is processed (step 609), the inbound data stream is transmitted to the host on the system I/O bus (step 611). The IOP then waits for the host's response (step 612), which is typically the next display screen. When a response is received, flow reverts to step 604. Typically, a data stream script for a workstation will include a think time command immediately after an AID key.

If the byte being parsed at step 605 is '00'X, a command is indicated. In this case, the command code byte (field 363) is examined. If the command is a think time command (step 620), the simulation program waits the specified think time period multiplied by the time multiplier specified in field 344 (step 621), increments the byte pointer by the length of the command (step 622), and flow goes back to step 605. If the command is any other command except an "End" command (step 623), the command is executed (step 624) and flow reverts to step 605. Although only the start, end and think time command are defined in the embodiment described herein, numerous other commands are possible and may be defined as part of the simulation protocol. For example, a "percent jump" command could be defined, in which a branch to another location in the data stream script is taken a specified percentage of the times that the command is executed, the decision whether to take the jump in any particular case being made by generating a random number as is known in the art. Such a command has the advantage of introducing random effects into the simulation, which may be desirable in certain circumstances. As another example, a "call" command could be defined, in which another data stream script is called and executed, control returning to the original data stream script when an "end" command is encountered in the called data stream script.

If an "end" command was indicated at step 623, the program verifies proper location of the end command, which should be at starting location plus offset (field 342) plus length of data stream script (field 341) minus 5 (the length of an end command), at step 625. If an error in end command location was indicated at step 611, the program branches to handle the error (step 630). If the end command is in the proper location, the data stream script has been successfully processed, and the program returns to executed the next step shown in FIG. 5.

In the preferred embodiment described herein, the data contained in the data stream script represents data received by the IOP from the workstation. This data is processed by the IOP just as it would process data from a real workstation. Portions of the IOP behave exactly as they would for a real workstation. Screen buffers and field format tables are maintained, and the control program processes keystrokes that look like real data, thus achieving a truer simulation. However, in an alternative embodiment, the data stream script could represent inbound data to the host, in which case little or no processing of the data would be required by the IOP. In the case of certain types of workstations and IOPs, there is little difference between the data the IOP receives from the workstation and the data it transmits to the host. For example, ASCII workstations would typically require very little processing of the data from the data stream script in the IOP. It will be recognized by those skilled in the art that the present invention could easily be adapted to IOPs servicing ASCII workstations (in either block mode or raw mode).

Because the IOP processor and memory function for the most part in their normal manner when processing simulated data, it is possible for an IOP to service one or more real interactive workstations concurrently with simulation of one or more workstations. The IOP will maintain a separate screen buffer and field format table in local dynamic memory 204 for each workstation, real or simulated. The control program will process data from each workstation, real or simulated, in the same manner. The only difference will be that data from a real workstation will appear on I/O device interface 207, while simulated data will be injected into the processor by that part of the control program executing the simulation script.

In the preferred embodiment described herein, the simulation script is separated from the IOP control program which executes it. The IOP control program is generic, capable of executing many different simulation scripts. Since the control program does not contain the

actual text of simulated data streams that will be sent to the host, it can be relatively compact as appropriate for storage in the limited space typically available in IOP local memory. Preferably, simulation scripts are stored on mass data storage, such as magnetic disk drives, until the simulation scripts are actually needed to run a simulation. When needed, a selected script will then be loaded into IOP local memory and executed as described herein. In an alternative embodiment, the text to be sent by the IOP to the host during simulation could be stored in the IOP local memory at all times, or could be stored in other storage devices available to the IOP. This text could be imbedded in a simulation control program or part of one or more separate simulation records.

In an additional alternative embodiment, it would not be necessary to maintain the control program portion responsible for executing simulation scripts in the IOP local memory during normal operation. In this embodiment, whenever a simulation is desired, a special simulation program could be loaded into local memory of the IOP, which may or may not contain the imbedded text of the data streams to be sent to the host to achieve simulation of I/O devices. This alternative would have the advantage of reducing the size of IOP local memory required to the extent of that part of the control program required for simulation execution, but would require additional complexity in the host operating system to support loading the different versions of IOP code, keeping track of which version is operating, etc.

The simulation script **301** is a record which may be created and transferred as any record in a computer system. It would, for example, be possible to create a simulation script with a general purpose file editor, although for large scripts this could prove somewhat tedious. In the alternative, a special purpose editor could be created which would provide interactive support for generating and editing the simulation script; e.g. default values for many of the simulation script fields could be provided automatically, field locations and lengths could be verified, etc., as is known in the art. As a further alternative, an application program could be created which would capture activity of a real I/O device during a sample interval and convert it to a simulation script.

Although a specific embodiment of the invention has been disclosed along with certain alternatives, it will be recognized by those skilled in the art that additional variations in form and detail may be made within the scope of the following claims.

What is claimed is:

1. A computer system comprising:

a central processing unit;

a system I/O bus coupled to said central processing unit; and

an I/O processor assembly coupled to said system I/O bus for communicating with one or more I/O devices attached to said I/O processor assembly via a communications path independent of said system I/O bus, said I/O processor assembly comprising a programmable processor for controlling the operation of said I/O processor assembly and a local memory for storing instructions which execute on said programmable processor;

wherein said I/O processor assembly has means for simulating an I/O device to said system I/O bus, said means comprising said programmable processor and said local memory, and further comprising:

(a) means for receiving a simulation script from said system I/O bus, and

(b) means for executing said simulation script to simulate an I/O device.

2. The computer system of claim **1**, wherein said I/O processor assembly has means for simulating a plurality of I/O devices simultaneously, said means for simulating a plurality of I/O devices simultaneously comprising said programmable processor and said local memory.

3. The computer system of claim **1**, further comprising:

a real I/O device coupled to said I/O processor assembly via a communications path independent of said system I/O bus;

wherein said means for simulating an I/O device comprises means for simulating an I/O device concurrently with servicing said real I/O device.

4. The computer system of claim **1**, wherein said simulation script comprises a plurality of data bytes representing simulated input data and a plurality of imbedded commands.

5. The computer system of claim **1**, wherein said means for executing said simulation script comprises a control program having a plurality of instructions stored in said local memory and which execute on said programmable processor.

6. A method for performing introspective tasks on a computer system, having a central processing unit, a system I/O bus, and an I/O processor assembly coupled to said system I/O bus, said I/O processor assembly for communicating with one or more I/O devices attached to said I/O processor assembly via a communications path independent of said system I/O bus, said method comprising the steps of:

receiving a command being transmitted on said system I/O bus to simulate an I/O device, said command being received by said I/O processor assembly;

generating a plurality of simulated data streams, said simulated data streams simulating data from said I/O device, said generating step being performed by said I/O processor assembly without assistance from a real I/O device; and

transmitting said simulated data streams on said system I/O bus, said transmitting step being performed by said I/O processor assembly.

7. The method for performing introspective tasks on a computer system of claim **6**, further comprising the step of:

receiving output destined for said I/O device being simulated from said system I/O bus, said output being received by said I/O processor assembly.

8. The method for performing introspective tasks on a computer system of claim **6**, wherein:

said command received by said I/O processor assembly commands said I/O processor assembly to simulate a plurality of I/O devices simultaneously; and

said generating and transmitting steps generate and transmit a plurality of simulated data streams, wherein said simulated data streams simulate data from said plurality of I/O devices simultaneously.

9. The method for performing introspective tasks on a computer system of claim **6**, further comprising the step of:

servicing, with said I/O processor assembly, a real I/O device coupled to said I/O processor assembly, wherein said servicing step is performed con-

currently with said steps of generating and transmitting said simulated data streams.

10. A method for simulating real-time performance of a hypothetical computer system having a first number of I/O devices with a real computer system having a second number of I/O devices attached to one or more I/O processor assemblies, wherein said first number is greater than said second number, comprising the steps of:

defining, with said real computer system, a plurality of I/O devices to be simulated by said real computer system;

issuing a command in said real computer system, said command directed to an I/O processor assembly of said real computer system, to simulate said I/O devices to be simulated;

simulating, with said I/O processor assembly, real-time behavior of said I/O devices to said real computer system,

wherein said steps of defining a plurality of devices, issuing a command and simulating real-time behavior are performed as part of a capacity planning task executed by said real computer system, in which information concerning operational characteristics of said hypothetical system is used to perform said capacity planning task.

11. The method of claim 10, wherein said step of issuing a command comprises sending a simulation script to said I/O processor assembly, and said simulating step comprises executing said simulation script with said I/O processor assembly.

12. The method of claim 11, wherein said simulation script comprises a plurality of data bytes representing simulated input data and a plurality of imbedded commands.

13. A method for simulating real-time performance of a hypothetical computer system having a first number of I/O devices with a real computer system having a second number of I/O devices attached to one or more I/O processor assemblies, wherein said first number is greater than said second number, comprising the steps of:

defining, with said real computer system, a plurality of I/O devices to be simulated by said real computer system;

issuing a command in said real computer system, said command directed to an I/O processor assembly of said real computer system, to simulate said I/O devices to be simulated;

simulating, with said I/O processor assembly, real-time behavior of said I/O devices to said real computer system,

wherein said steps of defining a plurality of I/O devices, issuing a command, and simulating real-time behavior are performed as part of a software development task executed by said real computer system to develop a software module, in which information concerning operational characteristics of said hypothetical system executing said software module is used to perform said software development task.

14. The method of claim 13, wherein said step of issuing a command comprises sending a simulation script to said I/O processor assembly, and said simulating step comprises executing said simulation script with said I/O processor assembly.

15. An I/O processor assembly for coupling to a system I/O bus of a computer system, and for communicating with one or more I/O devices attached to said I/O processor assembly via a communications path independent of said system I/O bus, said I/O processor assembly comprising:

a system I/O bus interface;

a programmable processor coupled to said system I/O bus interface for controlling operation of said I/O processor assembly;

a local memory for storing instructions which execute on said programmable processor, coupled to said programmable processor;

means for receiving a command being transmitted on said system I/O bus via said system I/O bus interface to simulate an I/O device attached to said I/O processor, said means for receiving a command including means for receiving a simulation script being transmitted on said system I/O bus; and

means, responsive to said means for receiving a command, for simulating said I/O device to said system I/O bus, said means for simulating said I/O device to said system I/O bus including means for executing said simulation script with said programmable processor.

16. The I/O processor assembly of claim 15, wherein said I/O processor assembly has means for simulating a plurality of I/O devices simultaneously, said means for simulating a plurality of I/O devices simultaneously including said programmable processor.

17. The I/O processor assembly of claim 15, wherein said means for simulating an I/O device comprises means for simulating an I/O device concurrently with servicing a real I/O device attached to said I/O processor assembly.

18. The I/O processor assembly of claim 15, wherein said simulation script comprises a plurality of data bytes representing simulated input data and a plurality of imbedded commands.

19. The I/O processor assembly of claim 15, wherein said means for executing said simulation script comprises a control program having a plurality of instructions stored in said local memory which execute on said programmable processor.

20. A computer system, said computer system having a first number of I/O devices attached to one or more I/O processor assemblies, said computer system comprising:

means for defining a hypothetical computer system having a second number of I/O devices, said second number being greater than said first number, wherein at least one of said second number of I/O devices is a device to be simulated in real-time by said computer system;

means for commanding an I/O processor assembly of said system to simulate said I/O device to be simulated;

means in said I/O processor assembly, responsive to said means for commanding an I/O processor assembly of said system to simulate said I/O device to be simulated, for simulating the real-time behavior of said simulated I/O device to said computer system, thereby simulating the real-time behavior of said hypothetical computer system.

21. The computer system of claim 20, wherein said means for commanding an I/O processor assembly to simulate an I/O device comprises means for transmitting a simulation script to said I/O processor assembly, and said means for simulating the real-time behavior of said simulated I/O device comprises means for executing said simulation script.

22. The computer system of claim 21, wherein said simulation script comprises a plurality of data bytes representing simulated input data and a plurality of imbedded commands.

* * * * *

US005357519A

United States Patent [19]

Martin et al.

[11] Patent Number: 5,357,519

[45] Date of Patent: Oct. 18, 1994

[54] **DIAGNOSTIC SYSTEM**

[75] Inventors: **Stephen R. Martin**, San Jose; **Randall O. Mooney, Jr.**, Boulder Creek, both of Calif.

[73] Assignee: **Apple Computer, Inc.**, Cupertino, Calif.

[21] Appl. No.: **771,127**

[22] Filed: **Oct. 3, 1991**

[51] **Int. Cl.**[5] **G01R 31/28**; G06F 11/00

[52] **U.S. Cl.** **371/15.1**; 371/16.1; 364/551.01

[58] **Field of Search** 371/15.1, 16.1, 20.1

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,489,414	12/1984	Titherly	371/15.1 X
4,694,408	9/1987	Zaleski	364/551
4,703,482	10/1987	Auger	371/16.1
4,760,329	7/1988	Andreano	371/15.1
4,810,958	3/1989	Mogi	371/15.1 X
4,831,560	5/1989	Zaleski	364/551.01
4,837,764	6/1989	Russello	371/16.1
4,953,165	8/1990	Jackson	371/16.1
5,157,665	10/1992	Fakhraie-Fard	371/15.1
5,168,216	12/1992	Dance	324/158 R
5,228,039	7/1993	Knoke	371/19

OTHER PUBLICATIONS

Product brochure on the Tech 1 modular diagnostic system (undated).

Product brochure on the Tech 1 brand "Scanner Cartridge," GM 81-91 ECM, part No. TK05020D (undated).

Product brochure on the Tech 1 brand "Scanner Cartridge," Ford 81-90 ECA, part No. TK02570B (undated).

Product brochure on the Tech 1 brand "Scanner Cartridge," Chrysler 83-90 EEC, part No. TK02580B (undated).

Product brochure on the Tech 1 brand "Scanner Cartridge," E/K 86-87 System, part No. TK03020 (undated).

Product brochure on the Tech 1 brand "System Cartridge," 88-91 Brake Cartridge, part No. TK03030A (undated).

Product brochure on the Tech 1 brand "System Cartridge," GM 88-91 Body Systems Cartridge, part No. TK03040A (undated).

Product brochure on the Tech 1 brand "System Cartridge," Hydra-Matic 88-91 Transmission Cartridge, part No. TK03590A (undated).

Product brochure on the Tech 1 brand "FTD Cartridges" (undated).

Product brochure on the Tech 1 brand "Special Function Cartridge," MAF Tester, part No. TK06090 (undated).

Product brochure on the Tech 1 brand "Interface Cartridge," RS232C I/F, part No. TK05030A (undated).

Product brochure on the Tech 1 brand "Accessories," Tech 1 Printer, part No. TK00540/TK00840 (undated).

Apple Computer, Inc., *Inside Macintosh*, vol. IV, (1986), pp. IV-238 through IV -295.

Apple Computer, Inc., *Inside Macintosh*, vol. V (undated but believed published in 1987), pp. V-347 through V-359 and V-573 through V-584.

Primary Examiner—Charles E. Atkinson

Assistant Examiner—Glenn Snyder

Attorney, Agent, or Firm—Blakely, Sokoloff, Taylor & Zafman

[57] **ABSTRACT**

A diagnostic apparatus for testing devices such as computer systems, and computer system components such as disk drives or printers. The device comprises a main unit, the main unit having a central processing unit for executing instructions, issuing commands, and receiving data from a first device. The apparatus also has a first peripheral unit coupled to the main unit, the first peripheral unit having ports for interfacing with the first device, the first peripheral unit being interchangeable with a second peripheral unit for interfacing with a second device. The apparatus also comprises a first non-volatile memory unit coupled to the main unit, the first non-volatile memory unit comprising a first set of tests for the first device, the first non-volatile memory unit being interchangeable with a second non-volatile memory unit comprising a second set of tests for a second device. These interchangeable parts are provided so that the user may test various types of hardware. The apparatus further comprises software means for providing termination on a bus and methods for simulating devices on a bus for remote entry into diagnostic programs.

3 Claims, 17 Drawing Sheets

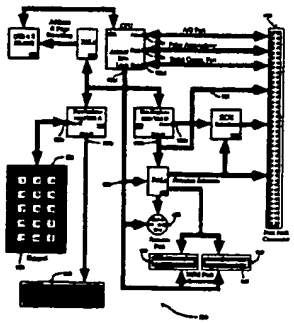


FIG 1

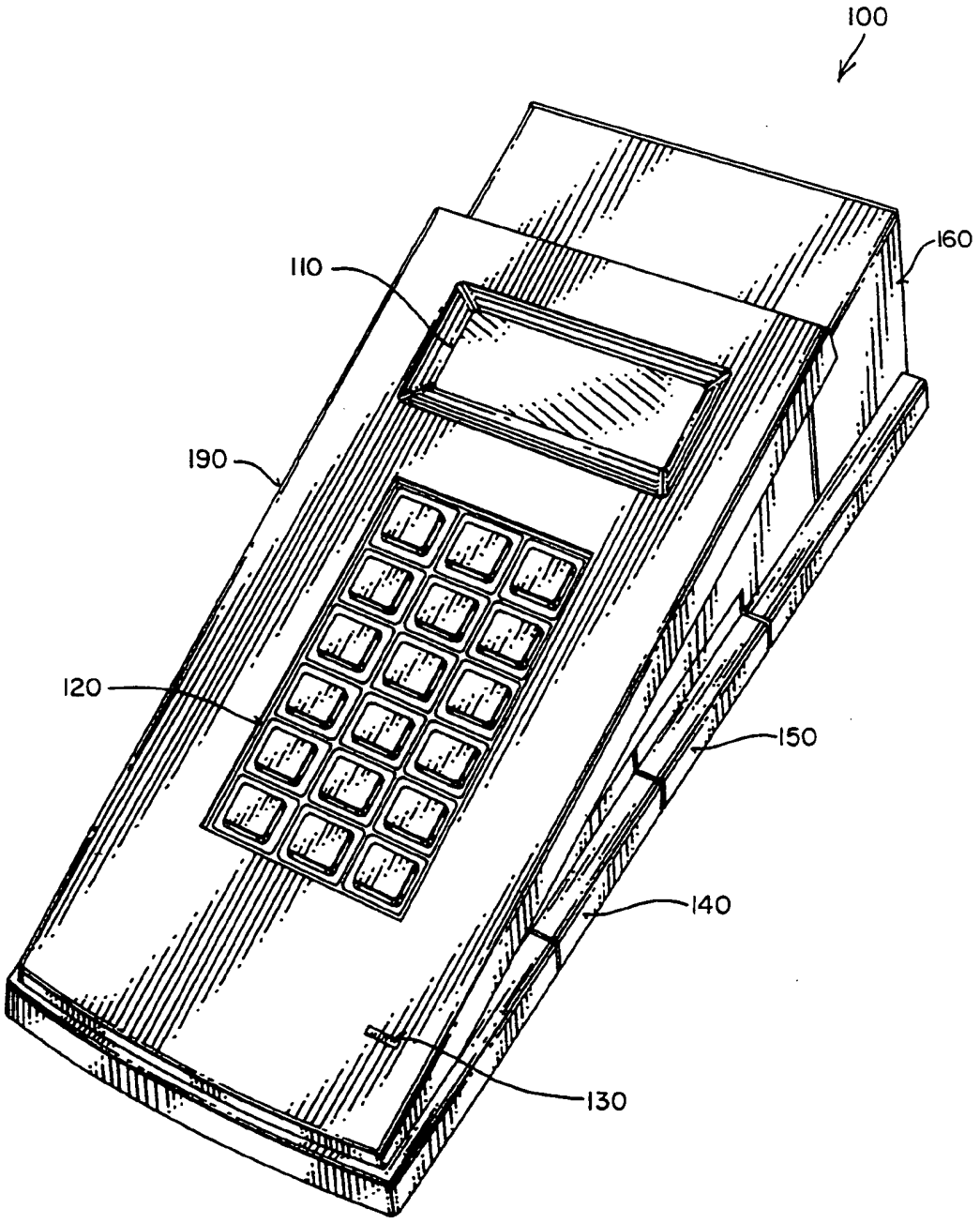
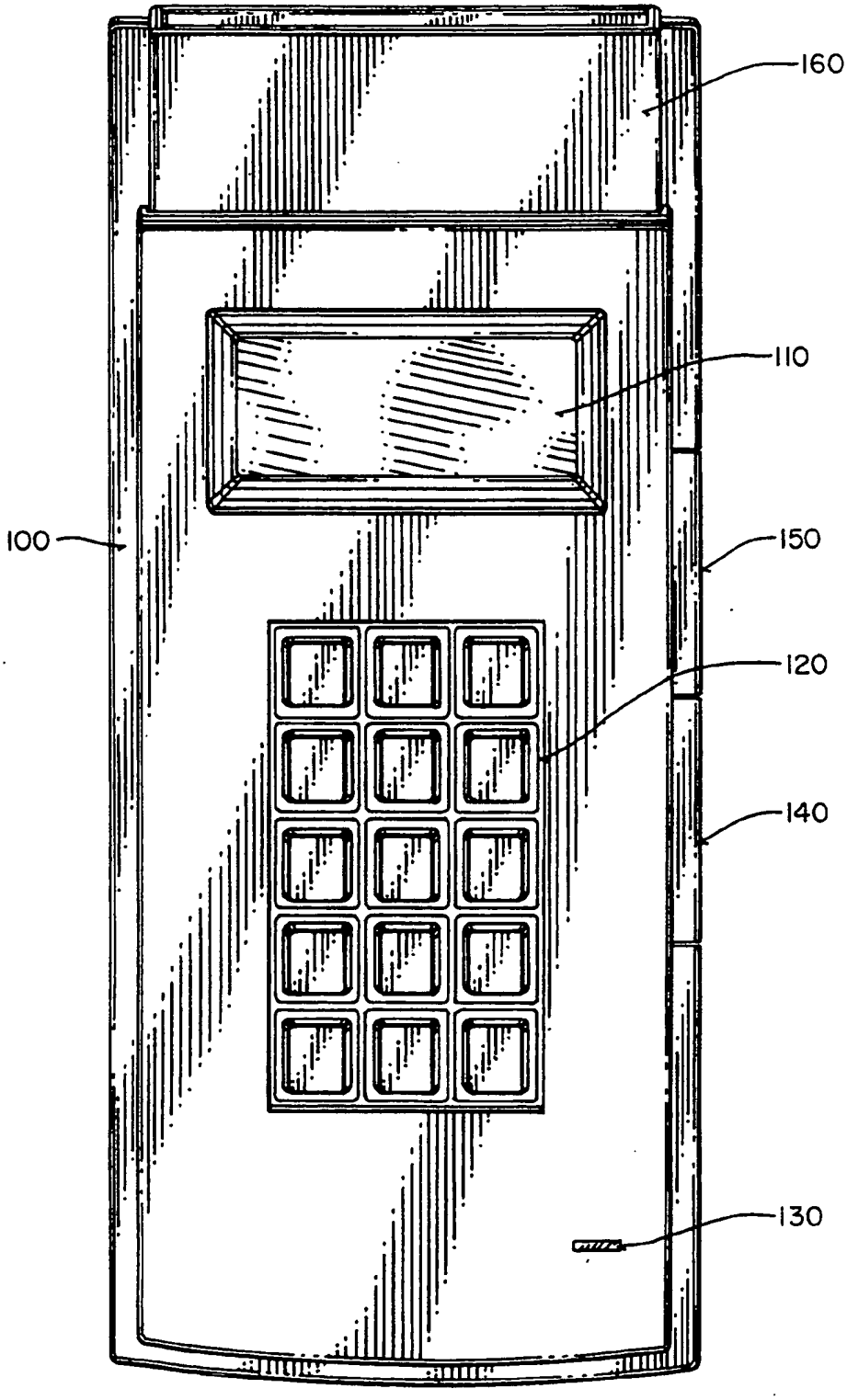


FIG 2



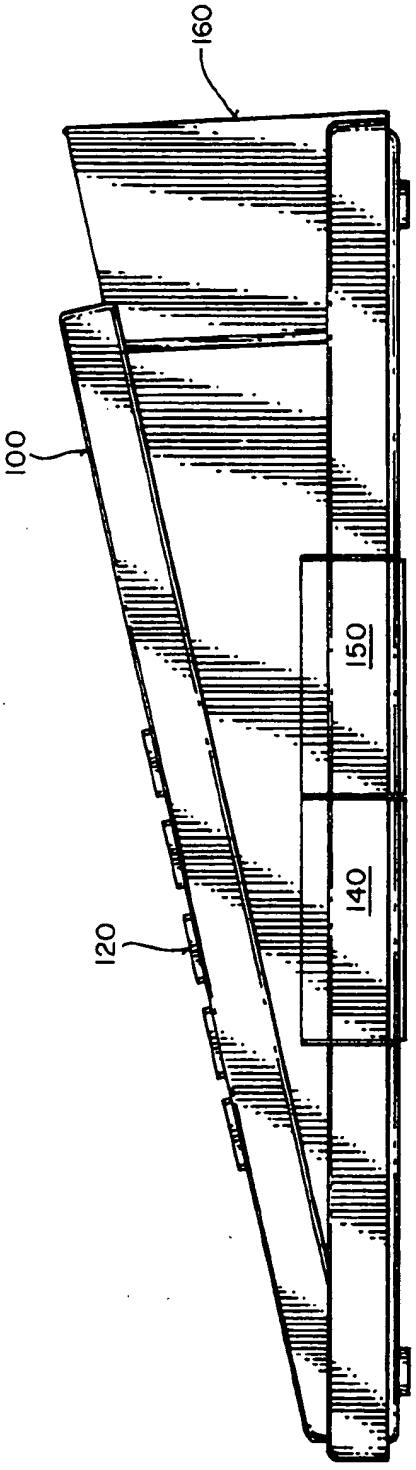


FIG. 3

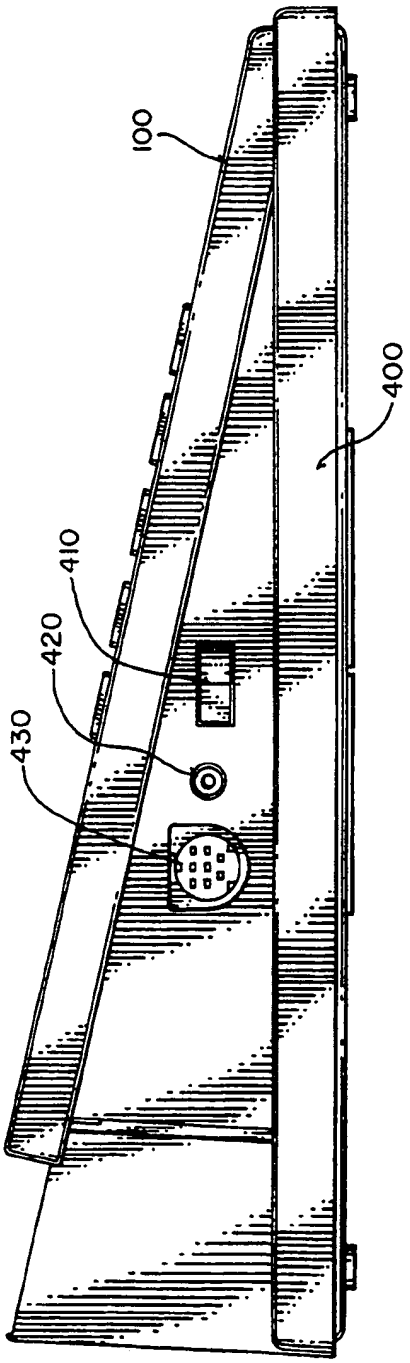


FIG. 4

FIG 5

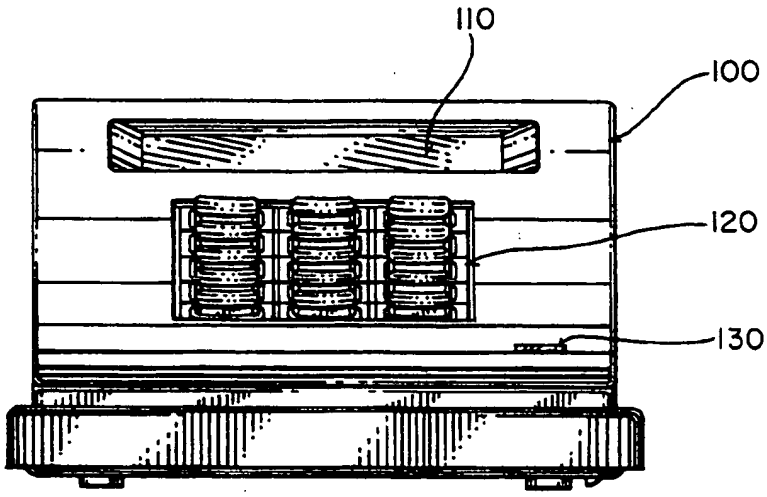
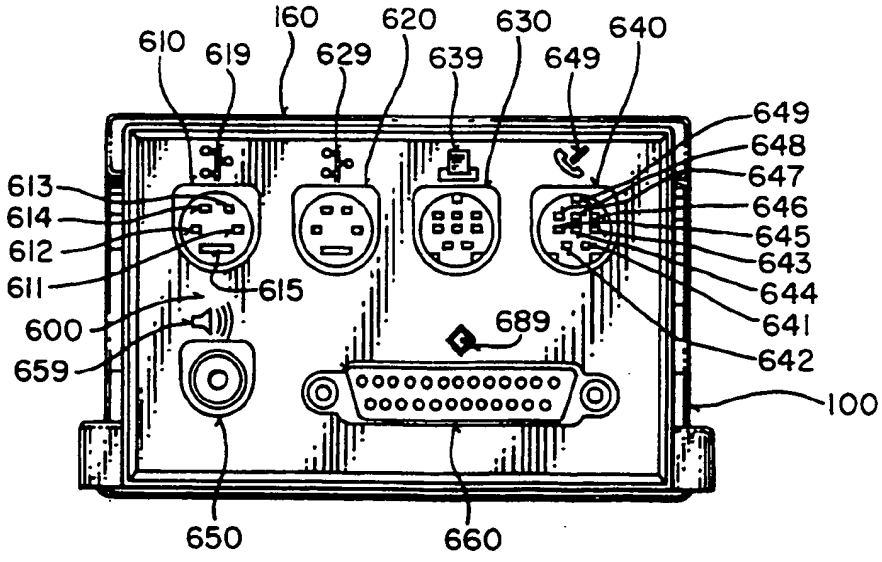

110
100
120
130

FIG 6


610
619
160
629
620
639
630
649
640
649
648
647
646
645
643
644
641
642
613
614
612
611
615
600
659
689
100
650
660

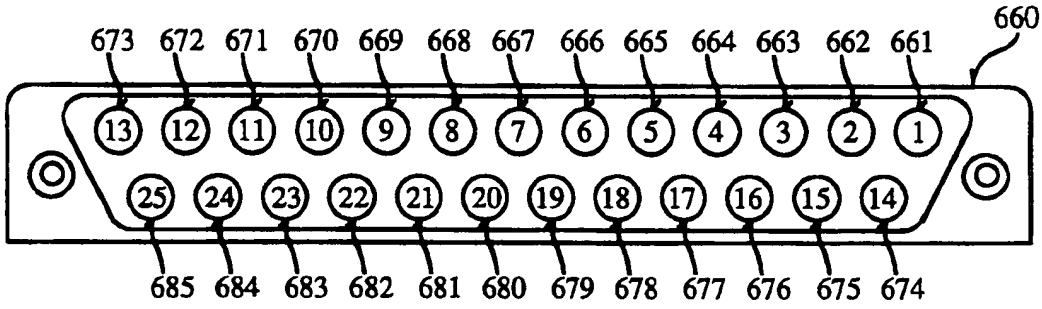

660
673 672 671 670 669 668 667 666 665 664 663 662 661
13 12 11 10 9 8 7 6 5 4 3 2 1
25 24 23 22 21 20 19 18 17 16 15 14
685 684 683 682 681 680 679 678 677 676 675 674

Figure 6a

FIG 7

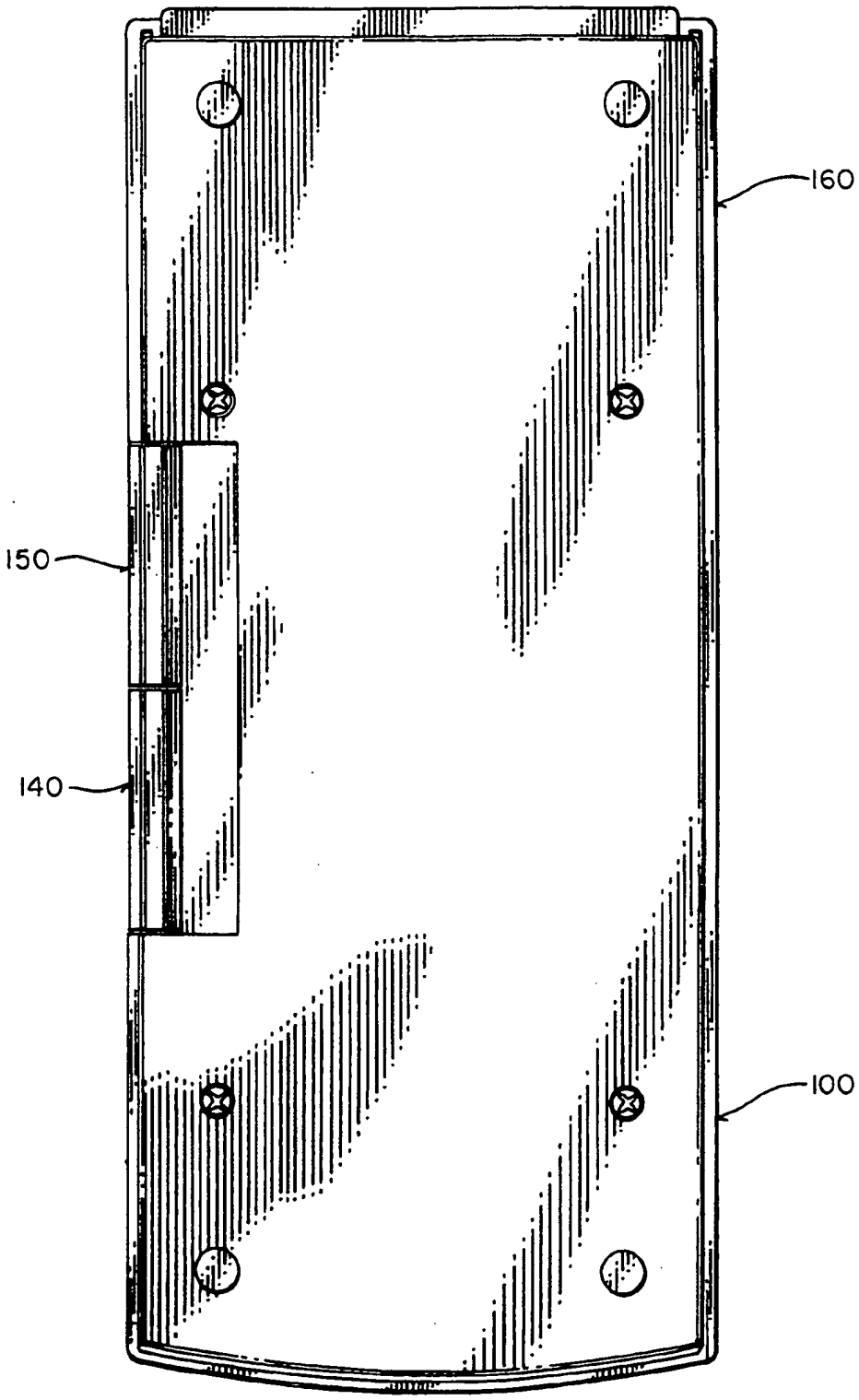
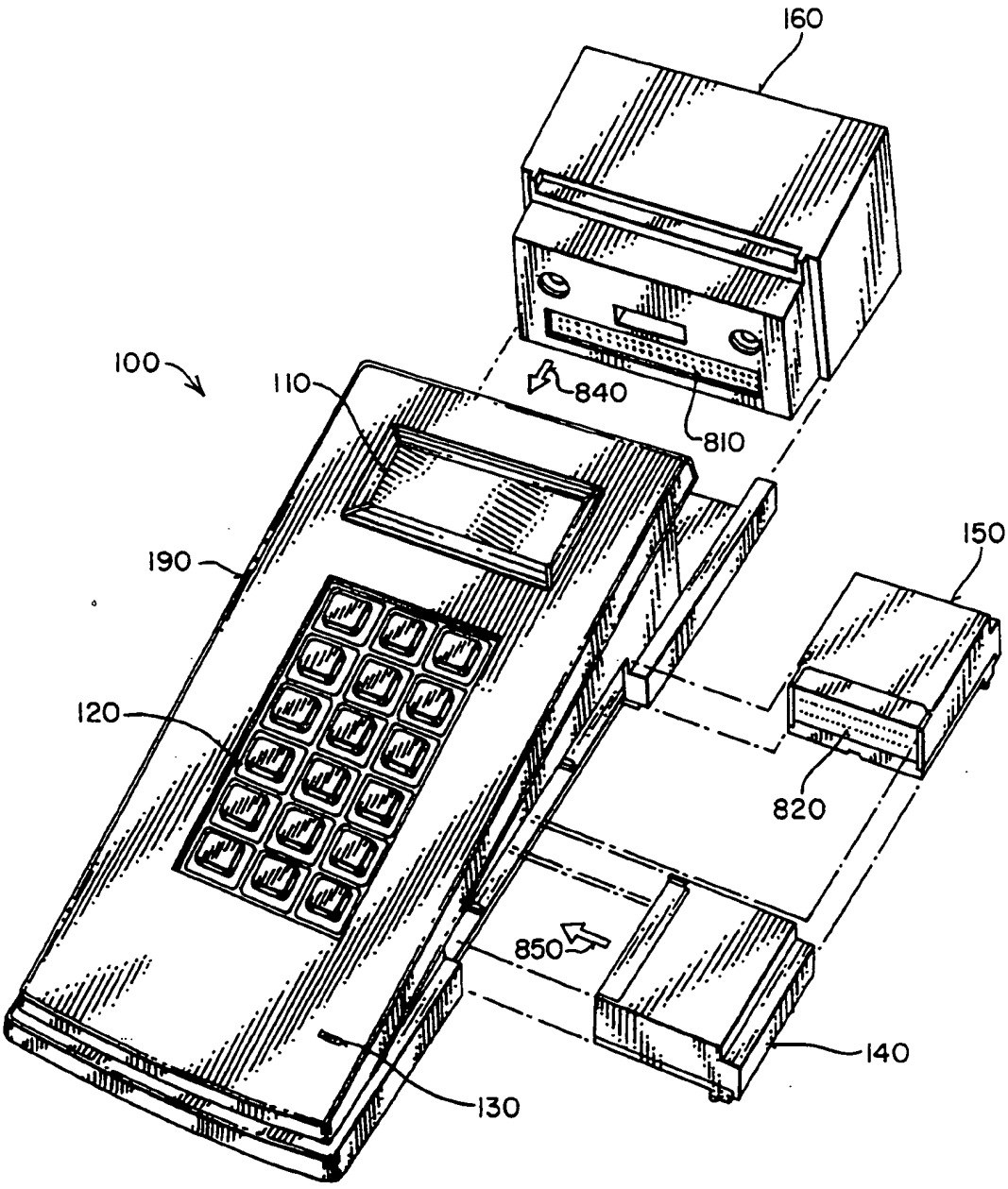


FIG _ 8



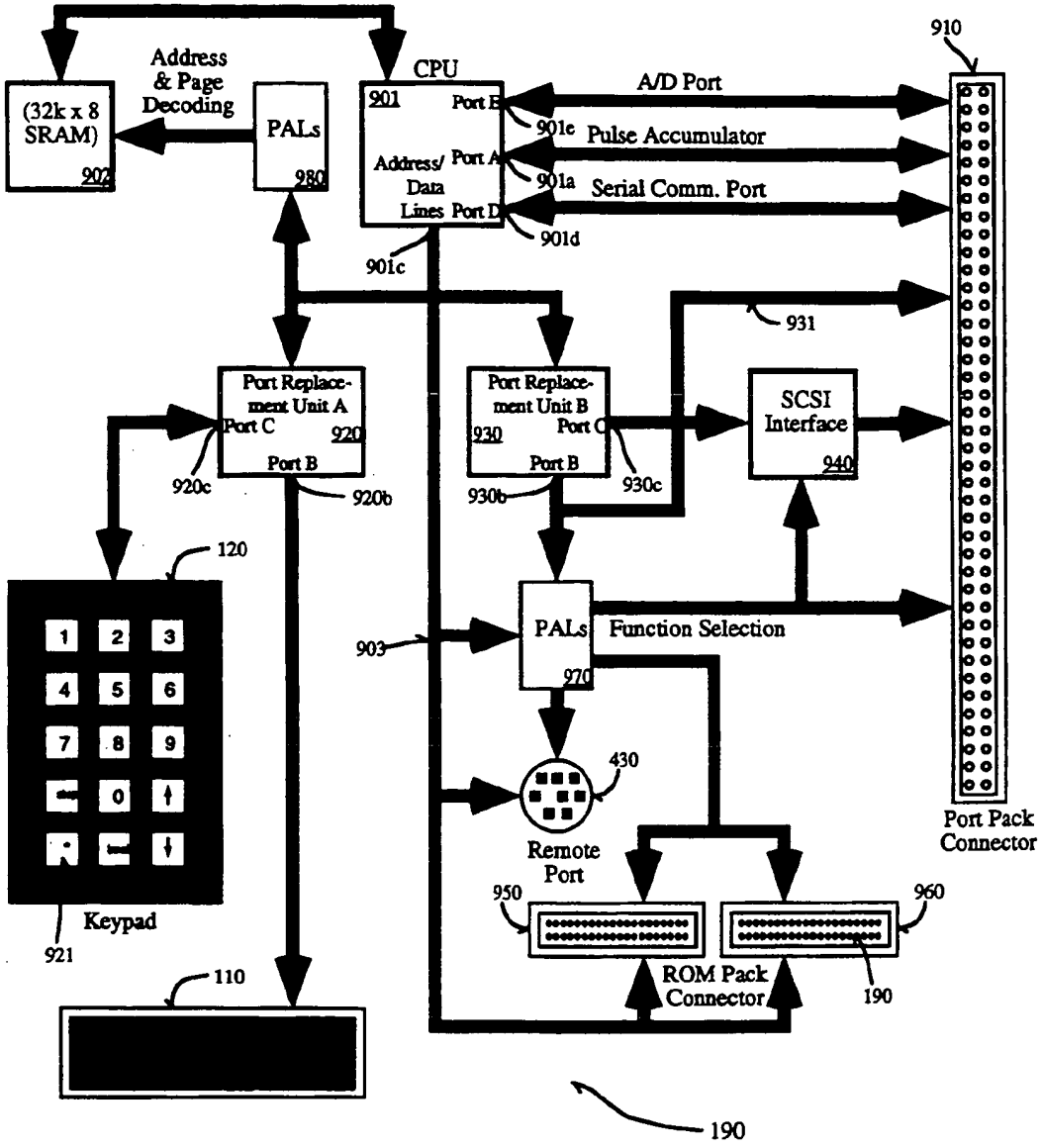


Figure 9 - Diagnostic Tool Base Unit

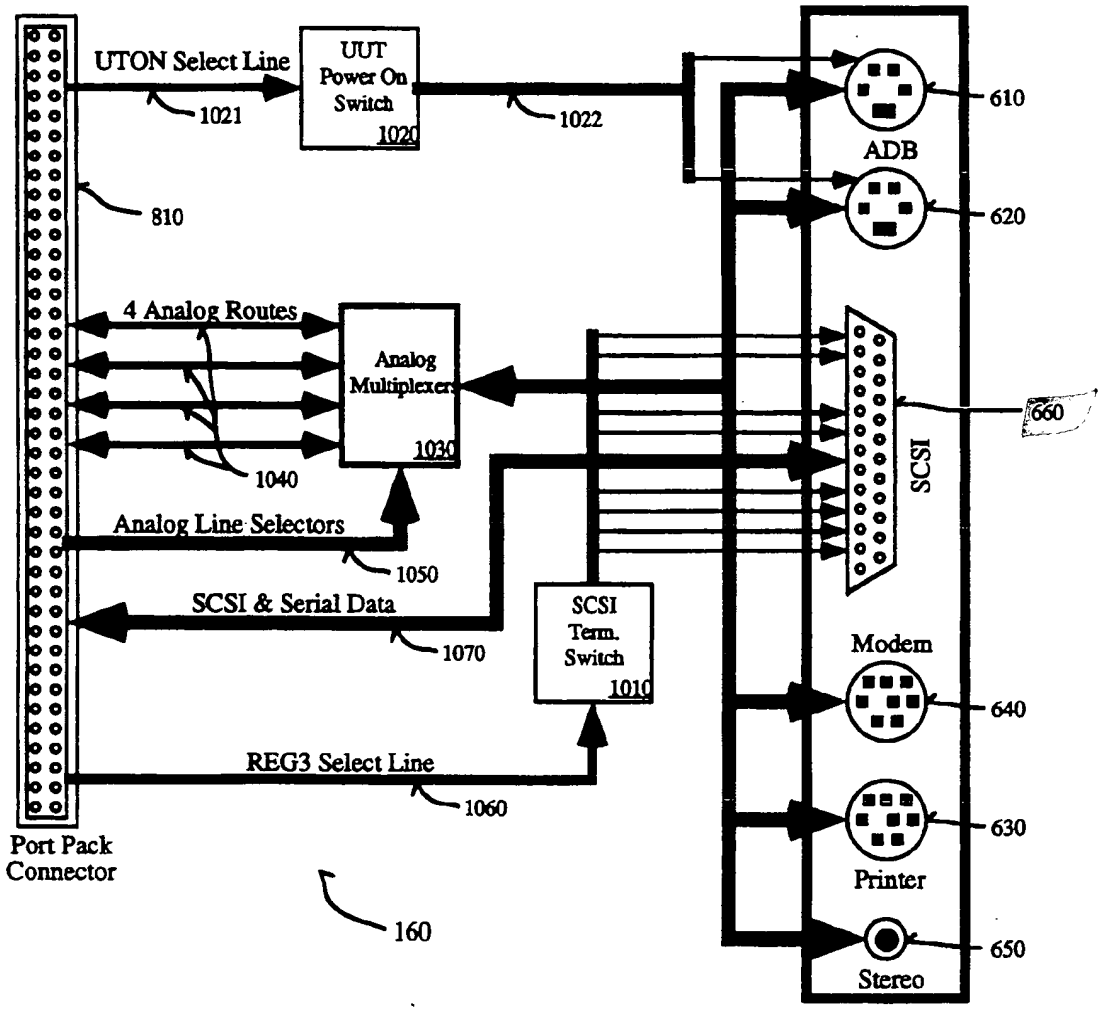


Figure 10 - Peripheral Port Pack

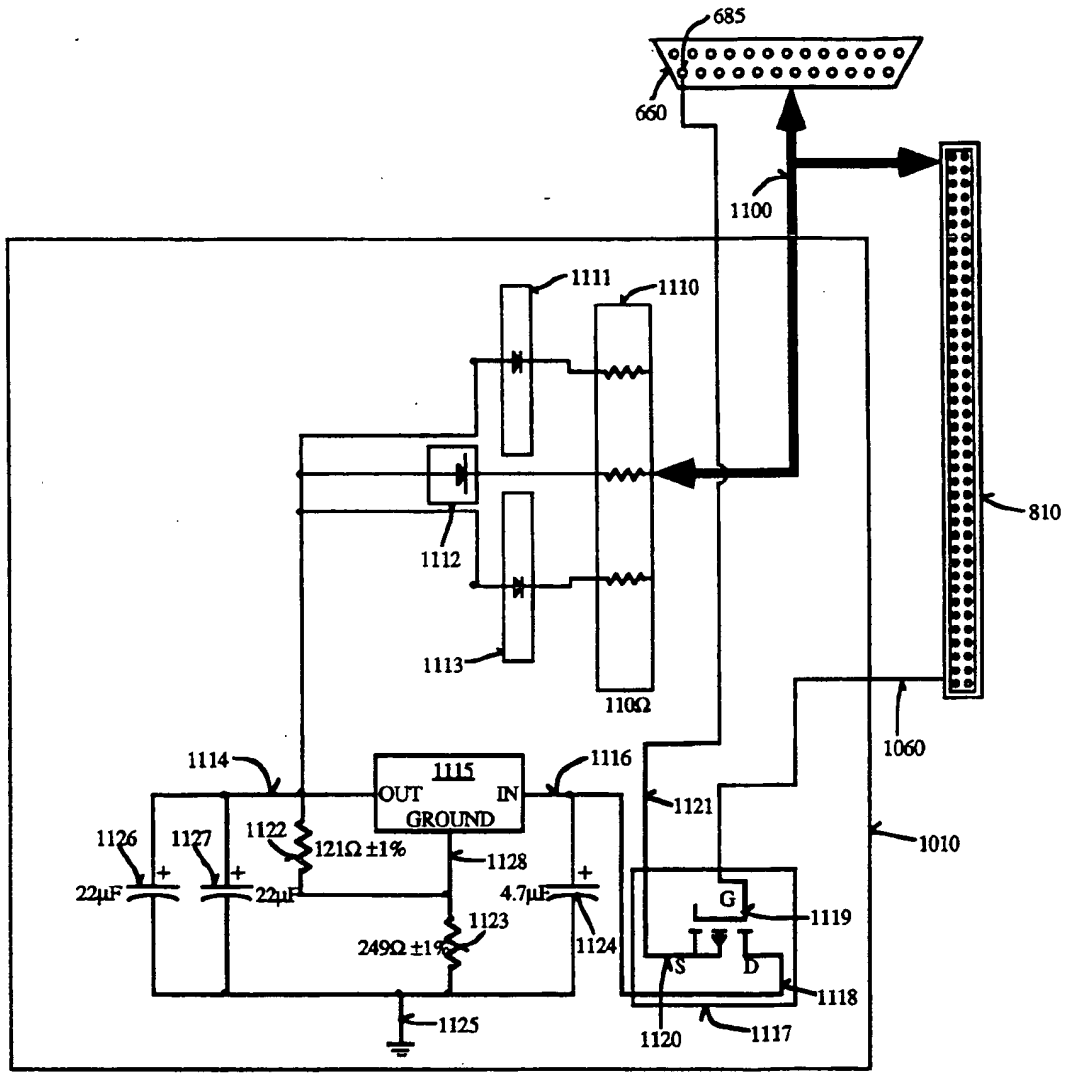

685
660
1100
1111
1110
810
1112
1113
110Ω
1060
1114
1115
OUT
IN
GROUND
1116
1121
1010
1126
1127
1122
121Ω ±1%
1128
22μF
22μF
4.7μF
1123
249Ω ±1%
1124
G
1119
S
D
1118
1125
1120
1117

Figure 11

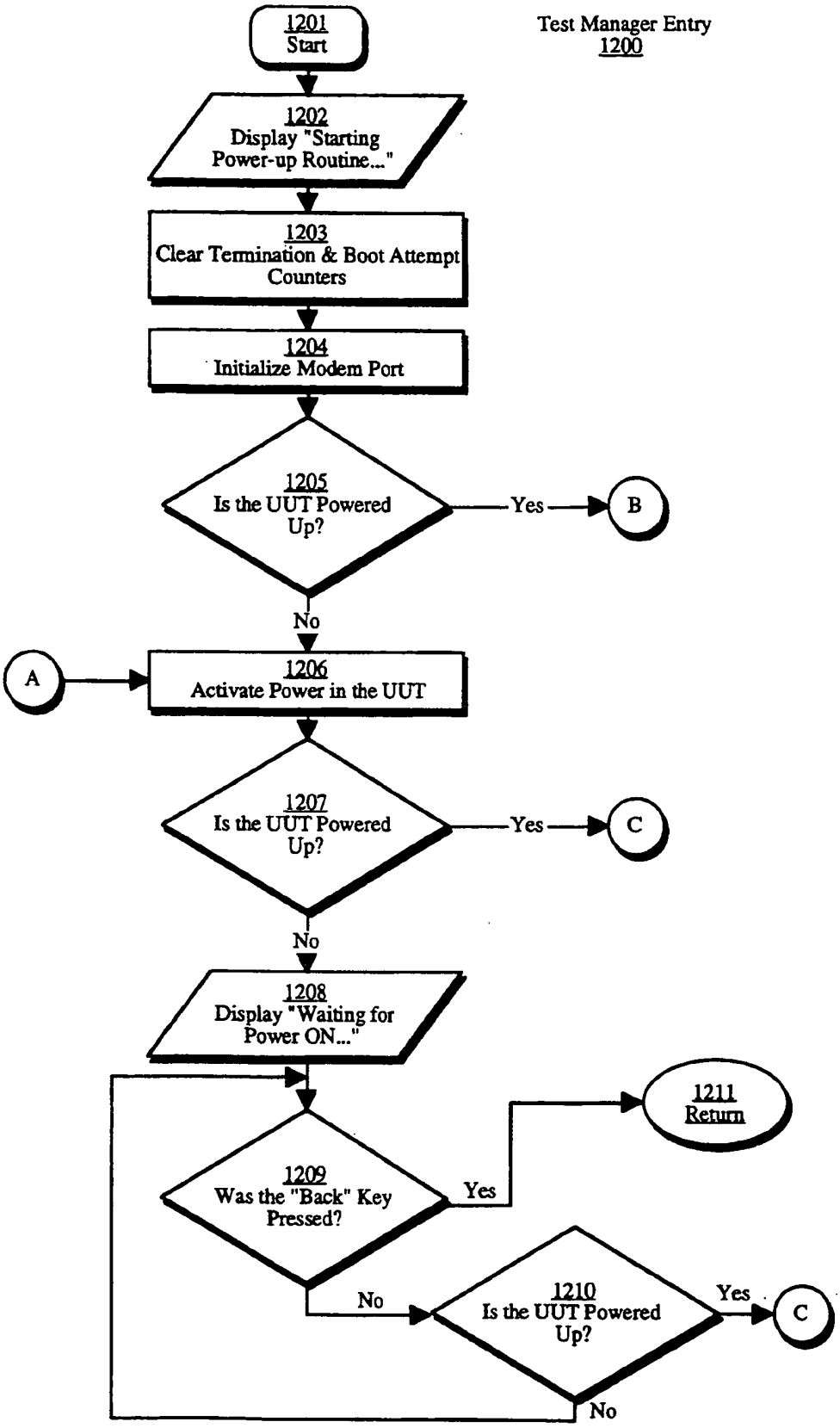

1201
Start
Test Manager Entry
1200
1202
Display "Starting Power-up Routine..."
1203
Clear Termination & Boot Attempt Counters
1204
Initialize Modem Port
1205
Is the UUT Powered Up?
Yes
B
No
A
1206
Activate Power in the UUT
1207
Is the UUT Powered Up?
Yes
C
No
1208
Display "Waiting for Power ON..."
1211
Return
1209
Was the "Back" Key Pressed?
Yes
No
1210
Is the UUT Powered Up?
Yes
C
No

Figure 12a

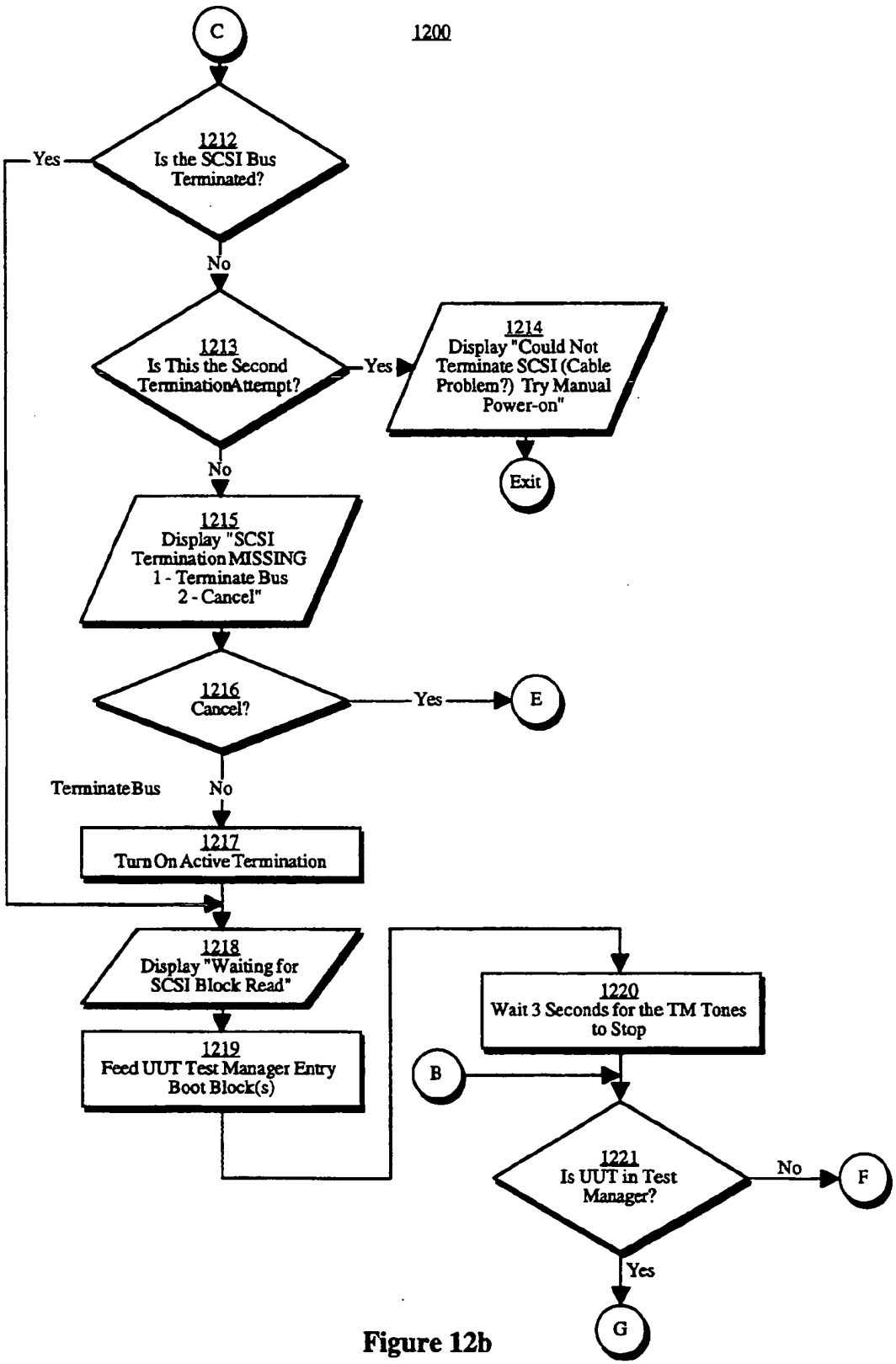

C
1200
1212
Is the SCSI Bus Terminated?
Yes
No
1213
Is This the Second TerminationAttempt?
Yes
1214
Display "Could Not Terminate SCSI (Cable Problem?) Try Manual Power-on"
Exit
No
1215
Display "SCSI Termination MISSING 1 - Terminate Bus 2 - Cancel"
1216
Cancel?
Yes
E
TerminateBus
No
1217
Turn On Active Termination
1218
Display "Waiting for SCSI Block Read"
1219
Feed UUT Test Manager Entry Boot Block(s)
1220
Wait 3 Seconds for the TM Tones to Stop
B
1221
Is UUT in Test Manager?
No
F
Yes
G

Figure 12b

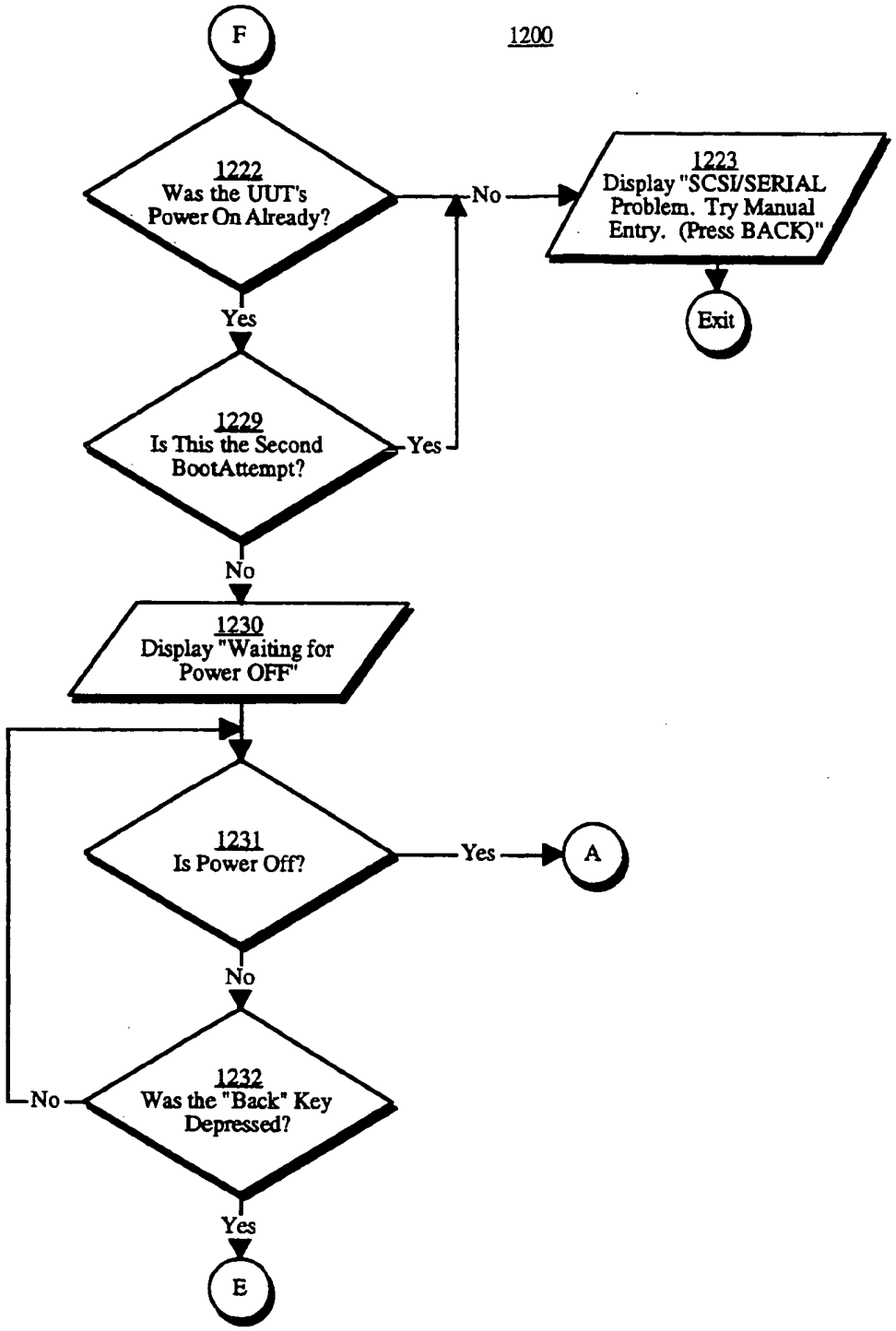

F
1200
1222
Was the UUT's Power On Already?
No
1223
Display "SCSI/SERIAL Problem. Try Manual Entry. (Press BACK)"
Exit
Yes
1229
Is This the Second BootAttempt?
Yes
No
1230
Display "Waiting for Power OFF"
1231
Is Power Off?
Yes
A
No
1232
Was the "Back" Key Depressed?
No
Yes
E

Figure 12c

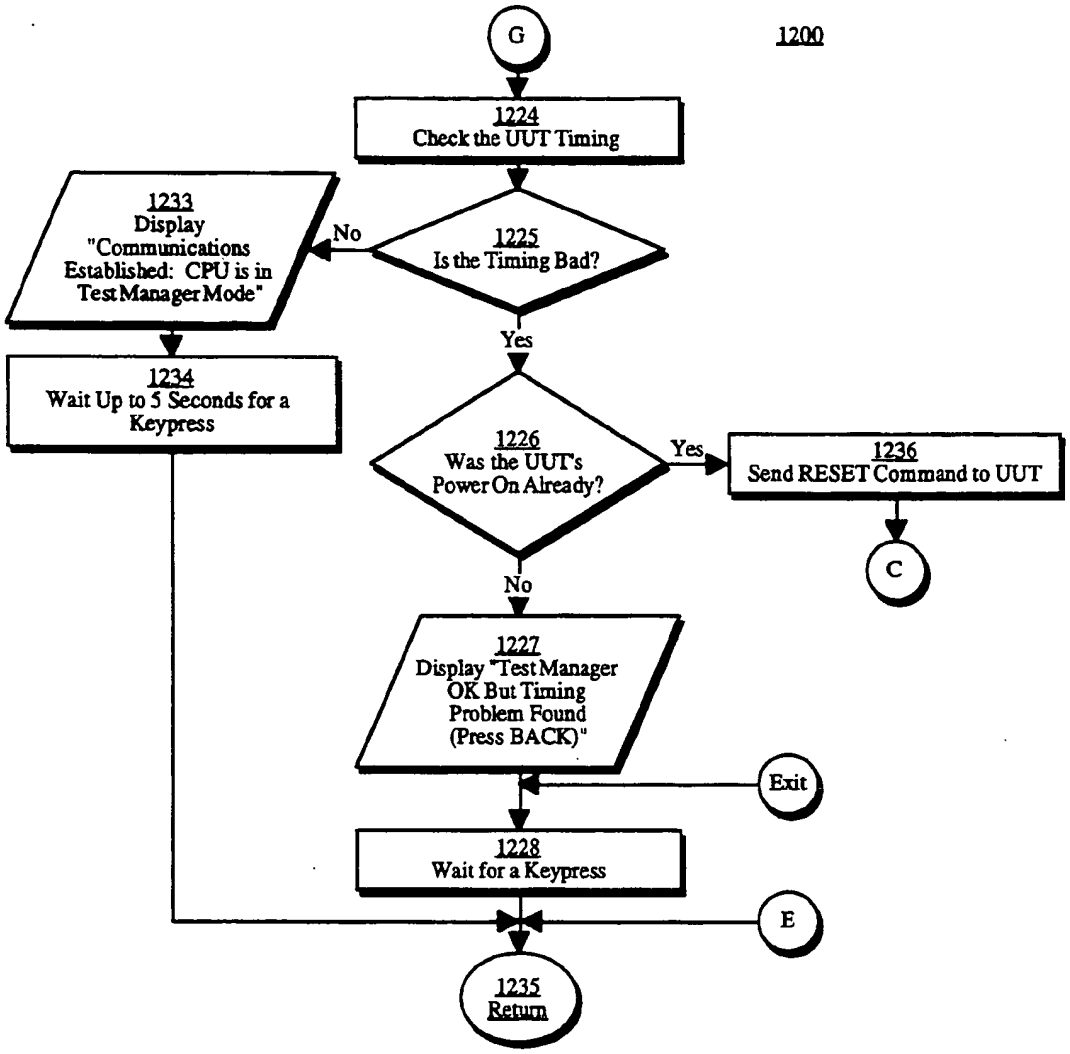
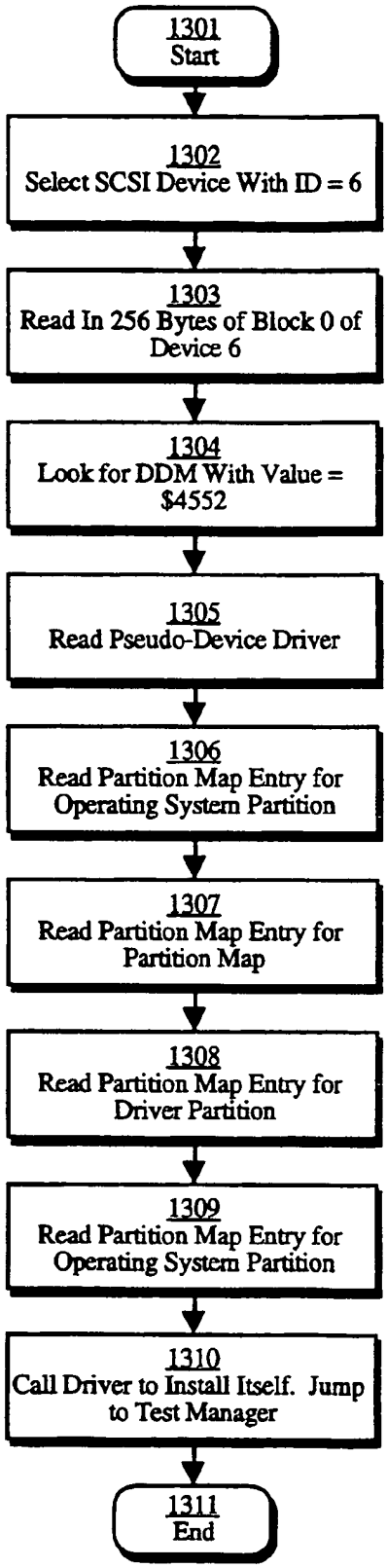

G
1200
1224
Check the UUT Timing
1225
Is the Timing Bad?
No
1233
Display
"Communications
Established: CPU is in
Test Manager Mode"
Yes
1234
Wait Up to 5 Seconds for a
Keypress
1226
Was the UUT's
Power On Already?
Yes
1236
Send RESET Command to UUT
C
No
1227
Display "Test Manager
OK But Timing
Problem Found
(Press BACK)"
Exit
1228
Wait for a Keypress
E
1235
Return

Figure 12d



SCSI TEST MANAGER STARTUP

1300

Figure 13

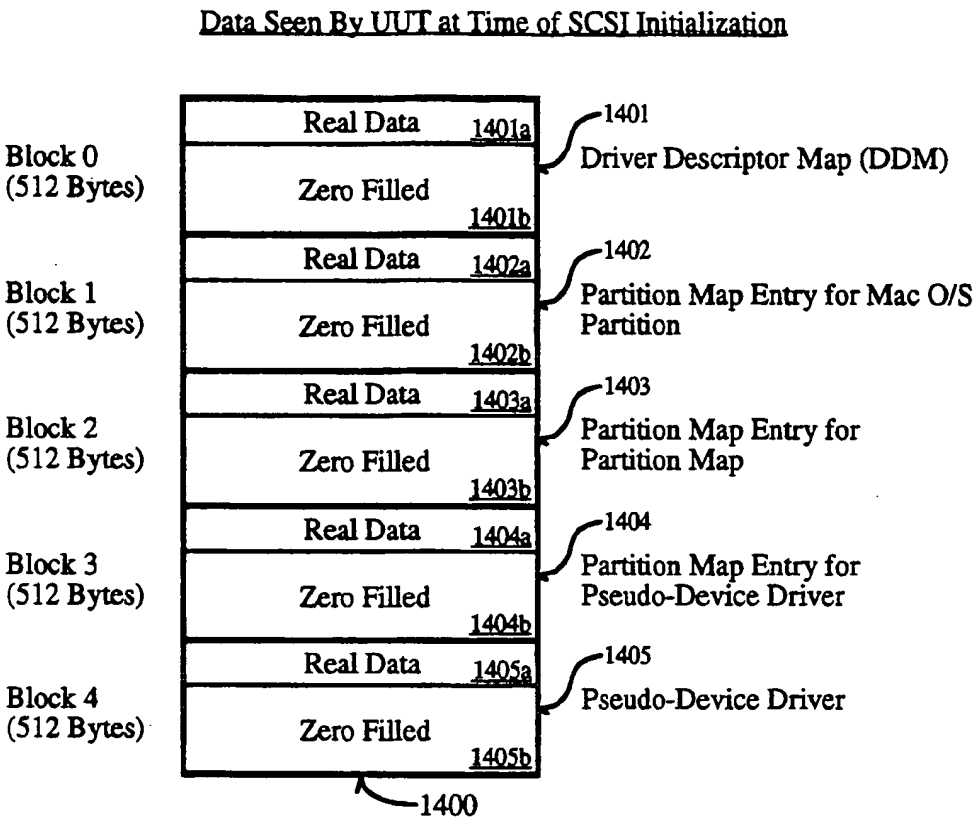

Data Seen By UUT at Time of SCSI Initialization
Block 0
(512 Bytes)
Real Data
1401a
Zero Filled
1401b
1401
Driver Descriptor Map (DDM)
Block 1
(512 Bytes)
Real Data
1402a
Zero Filled
1402b
1402
Partition Map Entry for Mac O/S Partition
Block 2
(512 Bytes)
Real Data
1403a
Zero Filled
1403b
1403
Partition Map Entry for Partition Map
Block 3
(512 Bytes)
Real Data
1404a
Zero Filled
1404b
1404
Partition Map Entry for Pseudo-Device Driver
Block 4
(512 Bytes)
Real Data
1405a
Zero Filled
1405b
1405
Pseudo-Device Driver
1400

Figure 14

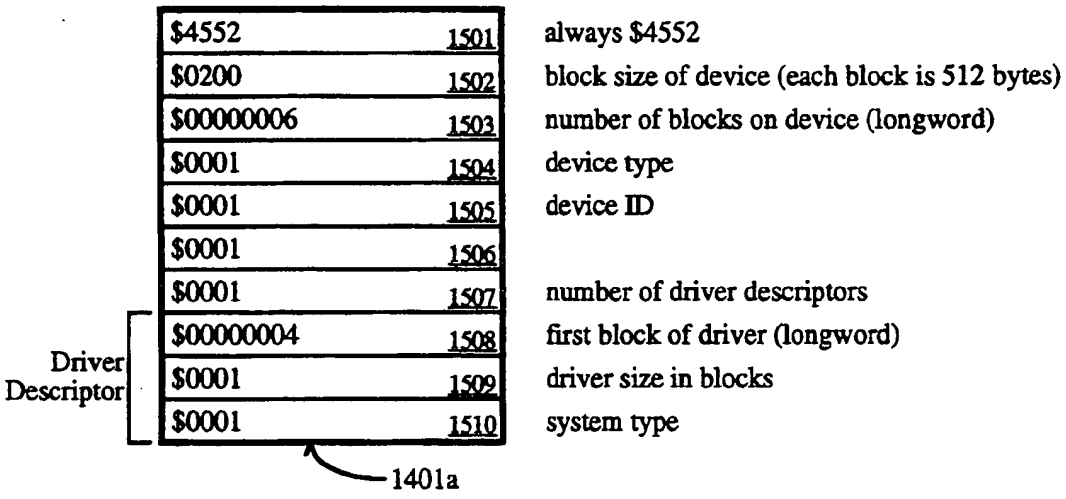


Figure 15 - Driver Descriptor Map

DIAGNOSTIC SYSTEM

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention relates to the field of diagnostic methods and apparatuses for computer systems. More specifically, this invention relates to a device and method for diagnosing a faulty computer system without disassembling the system.

2. Description of Related Art

The diagnosis of faulty computer systems, at times, can be a difficult process. For systems which are incapable of powering up and performing a system bootstrap initialization process, determining which components are faulty requires substantial time and effort using a variety of methods. One method of diagnosing a faulty computer system is to remove components in the computer system one by one and replace them with components which are known to work. Such a "trial and error" approach not only requires an on-hand inventory of components which are functional, but also requires that the computer system be disassembled. This type of diagnosis requires substantial effort to disassemble and reassemble the system and does not necessarily identify the underlying detect which caused the failure.

Yet another method for diagnosing a faulty computer system is to couple various types of diagnostic apparatuses to individual components. This process requires some disassembly of the computer system to attach diagnostic probes to the individual components. Depending on the location of individual components, this process may take a lot of time to disassemble the faulty device. This also requires specialized tools to test individual components.

For a system which is able to power up and execute a system bootstrap initialization but does not function properly, diagnosis may be performed using a software utility. Some types of utility programs may diagnose certain faulty components, but if the faulty components are required to operate the utility, the diagnosis of the system will be impossible. Even though the disassembly of the computer system may not be required to run such a diagnostic program, this program may be limited by the inability to test certain hardware components.

Some computer systems perform a software diagnostic self-test upon a system bootstrap initialization process. This type of routine tests various components in the system prior to operation to ensure that the system is operating properly. One such diagnostic routine is known as the "Test Manager" program which forms part of the bootstrap initialization procedure of the Macintosh® brand computer available from Apple Computer, Inc. of Cupertino, Calif. (Macintosh® is a trademark of Apple Computer, Inc.). This initialization process includes, among other things, initializing versatile interface adaptor (VIA) circuits, serial communication controller (SCC) circuits, floppy disk drive integrated circuit controllers, small computer system interface controller (SCSI) circuits, and sound device circuits coupled to the system. These routines, which are embedded in read-only memory (ROM)contained within the system, require that the system be capable of activating system power and initializing. If the system cannot be initialized, then certain types of tests may not be able to be performed.

Another drawback of prior approaches to diagnosing computer systems is that different computer systems

have a variety of interfaces and ports. Each port or interface requires a different connector and/or different circuitry to test these ports or interfaces. For instance, in addition to generic serial and parallel ports such as RS-232 standard ports or SCSI ports, some computer systems may have manufacturer-specific ports such as the Apple Desktop Bus (ADB) brand interface manufactured by Apple Computer, Inc. The wide variety of ports, interfaces and other coupling means for various systems makes it difficult to diagnose the many types of computer systems in the marketplace. Because the parameters of each interface and/or port must be known to the individual diagnosing the computer system, it is difficult for one person to diagnose a variety of machines. In addition, it is helpful if various types of connectors are available for coupling to the various systems for diagnosis. In summary, no single device possesses the necessary characteristics to diagnose various types of computer systems at all levels of operation.

SUMMARY AND OBJECTS OF THE INVENTION

One object of the present invention is to provide a device which can diagnose a computer system incapable of performing a system power up or bootstrap initialization process.

Another object of the present invention is to provide a device which performs nonintrusive diagnostics of a computer system.

Another object of the present invention is to provide a device which is capable of diagnosing various types of computer systems including system-specific connectors and interfaces.

Another object of the present invention is to provide a method and apparatus which facilitates diagnostics of a computer system with a minimal level of system-specific knowledge by a user.

These and other objects of the present invention are provided for by a diagnostic apparatus for testing devices such as computer systems, and computer system peripheral devices such as disk drives or printers. The apparatus comprises a main unit, the main unit having a central processing unit for executing instructions, issuing commands, and receiving data from a first device being tested. In a preferred embodiment, the apparatus comprises a display and keyboard for communicating with a user of the apparatus. The apparatus also has a first peripheral unit coupled to the main unit, the first peripheral unit having ports for interfacing with the first device, the first peripheral unit being interchangeable with a second peripheral unit for interfacing with a second device. The first peripheral unit may, for example, be used for testing one computer system such as a personal computer, and the second peripheral unit may be used for testing a second personal computer, disk drive, or printer. The apparatus also comprises a first non-volatile memory unit coupled to the main unit, the first non-volatile memory unit comprising a first set of tests for the first device. The first non-volatile memory unit is interchangeable with a second non-volatile memory unit comprising a second set of tests for a second device. These units are provided so that the user may test various types of hardware.

These and other objects of the present invention are provided for by a device for terminating a bus. In a preferred embodiment, the termination on a bus, for example a small computer system interface (SCSI) may

be desired. The device comprises a first means for activating termination of the bus which, in a preferred embodiment, is a flag in a register. When set, the flag switches on SCSI termination, and when not set (e.g., cleared) there is no termination. The device further comprises a second means coupled to the first means for supplying power, the second means being activated upon activation of the first means. In a preferred embodiment, the second means is a p-channel MOS-FET. The device also has a third means coupled to the second means for limiting voltage received from the second means which is a low dropout voltage regulator. Lastly, the device has a fourth means coupled to the third means for limiting transient voltages on the bus, which prevents the bus from appearing as if any device is coupled to the bus.

These and other objects of the present invention are provided for by a means for remotely entering a diagnostic program in a computer system. This method comprises simulating a first device coupled to the computer system and simulating a driver for the first device coupled to the computer system. The computer system is caused to load the driver and then to call the driver to execute itself in the computer system. The driver contains a jump instruction to the diagnostic program causing the diagnostic program to then execute.

BRIEF DESCRIPTION OF DRAWINGS

The present invention is illustrated by way of example and not limitation in the figures of the accompanying in which like references indicate like elements and in which:

FIGS. 1 through 7 show the external physical configuration of the diagnostic apparatus.

FIG. 8 shows a removable port pack and removable ROM packs in a disassembled configuration used in the diagnostic device of the preferred embodiment.

FIG. 9 is a block diagram of the base unit of the diagnostic device.

FIG. 10 is a block diagram of the port pack of the diagnostic device.

FIG. 11 is a schematic of the software controllable SCSI termination apparatus used by the diagnostic device.

FIGS. 12*a–d* is a flowchart showing various ways to enter test routines used in the diagnostic device of the preferred embodiment.

FIGS. 13 through 15 show the SCSI device emulation used by the preferred embodiment to enter the Test Manager brand diagnostic program.

DETAILED DESCRIPTION

A device and method for diagnosis of computer systems is described. In the following description, for the purposes of explanation, numerous specific details are set forth such as circuitry, signal names, signal lines, and part numbers are set forth in order to provide a thorough understanding of the invention. It will be obvious, however, to one skilled in the art that the invention may be practiced without these specific details. In other instances, well-known circuits, structures, and techniques have not been shown in detail in order to not unnecessarily obscure the present invention.

Physical Configuration of the Diagnostic Tool

FIG. 1 shows the external physical configuration of the preferred embodiment of the present invention. Diagnostic device **100** is generally used for testing the operation of computer systems, however, it may be used for testing individual components of computer systems such as hard disk drives, printers, monitors, or other peripherals. Diagnostic device **100** comprises several portions which are shown in their assembled and operating configuration in FIG. 1. Diagnostic device **100** comprises a display **110** for presenting information to the user of the device. Display **110** is one of the many liquid crystal displays (LCD's) which are commercially available and well-known to those skilled in the art. Diagnostic device **100** further comprises a keypad **120** for indicating command selections and other information to device **100**. Further device **100** comprises a low-power light emitting diode (LED) **130** indicating when battery power in the unit is becoming depleted. In order to provide the utmost flexibility, device **100** comprises modules **140, 150,** and **160** which are all removable from device **100** and interchangeable with other modules. **140** and **150** are removable "ROM packs" which provide different tests and different operating modes for the various computer systems or computer peripherals (hereinafter units under test or UUT's) which may be tested by device **100**. **160** is also a removable module and is known as a "port pack." Port pack **160** provides computer system ports for the various types of ports which may be present on a system being tested. ROM packs **140** and **150** and port pack **160** will be discussed in more detail below.

FIG. 4 shows one side **400** of device **100**. Side **400** of device **100** comprises a power switch **410** which is a single pole, single throw (SPST) rocker switch which is used to activate power to device **100**. Further, device **100** comprises an adaptor plug **420** which is a miniature dual conductor plug used for supplying 5 volt power to unit **100** via an integrated power supply adaptor cable. Diagnostic device **100** may also be powered by an internal battery when not coupled to a power supply via plug **420**. In addition, as shown on side **400** of FIG. 4, device **100** comprises a female serial port **430** which is used for coupling device **100** to a modem (modulator/-demodulator) for communication over telephone lines. This provides the capability for device **100** to communicate with other devices **100** or UUT's which can be remotely tested. Connector **430** is a mini 8-pin serial port connector which conforms to EIA standard RS-232 signal conventions. The signals and pins on connector **430** are described in more detail in *Guide to the Macintosh Family Hardware*, Second Edition by Apple Computer, Inc. available from Addison-Wesley Publishing Company, Inc. (copyright 1990)(hereinafter "Hardware Guide") at pages 357–374. Various ports and detailed connections to port **430** will be discussed in more detail below.

Removable ROM Packs

For the greatest flexibility in testing various types of computer systems and other devices with diagnostic tool **100**, ROM packs **140** and **150** and port pack **160** are provided which are all removable from main unit **190** of device **100** as shown in FIG. 8. These packs may be replaced with other packs which have different ports and/or different of tests. As is shown in FIG. 8, port pack **160** may be coupled to main unit **190** via connector **810**. Connector **810**, in the preferred embodiment, is a Fujitsu FCN215Q080-G/O, 80-pin male connector. A detailed description of the signals and the lines on connector **810** will be discussed below **810** provides an interface with main unit **190** of device **100** so that com-

munication with various peripheral ports on pack **160** may be accomplished. Port pack **160** may be coupled to main unit **190** by affixing **160** to **190** in the direction shown as **840** on FIG. **8**. This mates connector **810** with a corresponding female connector on main unit **190** (not shown). In addition, **100** comprises removable ROM packs **140** and **150** which comprise non-volatile test routines and other device or system-specific diagnostic programs. As is shown in FIG. **8**, each module such as **150** comprises a connector such as **820** which is mated to the main unit **190** to provide access to the test routines embedded in the non-volatile memory of each ROM pack **140** or **150**. Each ROM pack such as **140** and **150** may be coupled to main unit **190** via a 40-pin Fujitsu FCN215Q040-G/O male connector which mates with a corresponding female connector on main unit **190** (not shown). A ROM pack such as **140** may be inserted into main unit **190** in a direction shown by arrow **850** in FIG. **8** to provide communication between main unit **190** and ROM packs **140** and **150**.

Removable Port Pack

A more detailed view of the rear **600** of port pack **160** is shown in FIG. **6**. **600** of **160** shows various ports contained in one port pack **160** which is designed for testing various models of the Macintosh brand computer system family manufactured by Apple Computer, Inc. of Cupertino, Calif. Port pack **160** is used for testing the models Macintosh SE, SE/30, and the Macintosh II brand family of computers including the IIx. IIcx, among others. It can be appreciated by one skilled in the art, however, that a port pack other than **160** may be plugged into base unit **190** of diagnostic device **100** in order to provide a different set of ports. In the preferred embodiment, port pack **160** has six ports. Base unit **190** may accommodate any number of ports depending on the configuration of the port pack. It will be appreciated by one skilled in the art, that any number of ports may be present on a peripheral port pack in various embodiments of the present invention. A detailed description of the ports available on port pack **160** of device **100** shown in FIG. **6** will now be discussed.

As is shown in FIG. **6**, side **600** of port pack **160** comprises several ports which interface with Macintosh brand computer systems. Port pack **160** comprises two Apple Desktop Bus (ADB) brand connectors **610** and **620**, two miniature 8-pin EIA RS-424 standard serial connectors **630** and **640**, a two-conductor miniature audio connector **650** for coupling to audio ports, and a 25-pin female DB25 SCSI connector **650** for coupling to devices such as SCSI disk drives and disk drive ports on computer systems. ADB ports **610** and **620** are used for coupling with ADB brand ports on Macintosh computer systems for communicating various information to and from the computer via devices such as keyboards, trackballs, or mice. The pin assignments for **611** through **614** are shown in FIG. **6** and the corresponding signals are described with reference to Table 1 below.

TABLE 1

Signal Assignments for ADB Connector 610

Pin Number	Signal Name	Signal Description
611	ADB	Bidirectional data bus used for input and output. It is an open-collector signal pulled up to +5 V through a 470 Ω resistor on an Apple Macintosh brand computer's main logic board.
612	POWER.ON	On the Macintosh II family, a key on the keyboard momentarily grounds this pin to pin 614 to switch on the power supply. On other Apple Macintosh brand computers this pin is not connected.
613	+5 V	+5 volts.
614	GND	Ground.

A more detailed discussion of the ADB signals and devices used in the preferred embodiment is discussed in *Hardware Guide* at pages 287–326. A more detailed discussion of signals issued by device **100** for performing diagnostics of a computer system is discussed below. The pin assignments for connector **620** are the same as those used on connector **610**. **610** are **620** are indicated by their corresponding labels **619** and **629** shown in FIG. **6**, the ADB icons. Further, each of the connectors has a key **615** associated with it such as shown with reference to **610**, to prevent improper insertion of cables into connectors **610** or **620**.

Port pack **160** further comprises two serial ports **630** and **640**. Serial ports **630** and **640** are used for coupling to a printer port or a modem port on a system as indicated by the appropriate labels **639** or **649**. As discussed above, each of the serial ports in the preferred embodiment and on port pack **160** conform to the EIA RS-422 standard signal conventions which are described in more detail in *Hardware Guide* at pages 357–374. The pin assignments for serial port **640** is described with reference to Table 2 below. The signal assignments for the pins on **630** are the same.

TABLE 2

Signal Assignments for Mini 8-pin Serial Port Connector 640

Pin Number	Signal Name	Signal Description
641	HSKo	Handshake output. Driven inverted. Voh = 3.6 V; Vol = −3.6 V; Ri = 450Ω
642	HSKi	Handshake input or external clock. Received uninverted. Vih = 0.2 V; Vil = −0.2 V; Ri = 12KΩ
643	TxD−	Transmit data (inverted). Driven inverted or tristated depending on the operation mode. Voh = 3.6 V; Vol = −3.6 V; Ri = 450Ω
644	GND	Signal ground. Connected to logic and chassis ground.
645	RxD−	Receive data (inverted). Vih = 0.2 V; Vil = −0.2 V; Ri = 12KΩ
646	TxD+	Transmit data. Driven uninverted or tristated depending on the operation mode. Voh = 3.6 V; Vol = −3.6 V; Ri = 450Ω
647	GPi	General-purpose input. Vih = 0.2 V; Vil = −0.2 V; Ri = 12KΩ
648	RxD+	Receive data. Received uninverted. Vih = 0.2 V; Vil = −0.2 V; Ri = 12KΩ

The two remaining ports on port pack **160** of the preferred embodiment are shown as **650** and **660** in FIG. **6**. **650** is a standard 2-conductor female monaural audio miniature jack used for diagnosing the sound capabilities of the device being tested. This is indicated by the sound label icon **659** above the sound jack. The processing of the signals received from sound jack **650** is discussed below.

Lastly, **160** comprises SCSI connector **660** which is a standard 25-pin female DB25 SCSI connector which is discussed in more detail in *Hardware Guide* at pages

375–396. SCSI connector **660** is labelled by icon **669** as shown in FIG. **6**. The circuitry coupled in main unit **190** to port pack **160** and thus to connector **660** is the discussed in more detail below. Pin outs on connector **660** are assigned as shown in FIG. **6***a* with reference to Table 3 below.

TABLE 3

Signal Assignments for SCSI Connector 660

Pin Number	Signal Name	Signal Description
661	/REQ	Request for a REQ/ACK data transfer handshake
662	/MSG	Indicates the message phase
663	/I/O	Controls the direction of data movement
664	/RST	SCSI data bus reset
665	/ACK	Acknowledge for a REQ/ACK data transfer handshake
666	/BSY	Indicates whether SCSI data bus is busy
667	GND	Ground
668	/DB0	Bit 0 of SCSI data bus
669	GND	Ground
670	/DB3	Bit 3 of SCSI data bus
671	/DB5	Bit 5 of SCSI data bus
672	/DB6	Bit 6 of SCSI data bus
673	/DB7	Bit 7 of SCSI data bus
674	GND	Ground
675	/C/D	Indicates whether control or data is on the SCSI bus
676	GND	Ground
677	/ATN	Indicates an attention condition
679	/SEL	Selects a target or an initiator
680	/DBP	Parity bit for SCSI data bus
681	/DB1	Bit 1 of SCSI data bus
682	/DB2	Bit 2 of SCSI data bus
683	/DB4	Bit 4 of SCSI data bus
684	GND	Ground
685	TPWR	+5 volts terminator power

Circuitry of the Diagnostic Tool

A block diagram of the main unit of diagnostic apparatus **190** is shown with reference to FIG. **9**. Central processing unit **901** of diagnostic tool **100** is a 68HC11E9 microcontroller manufactured by Motorola, Inc. of Schaumburg, Ill. The HC11E9 version of the microcontroller comprises the following features which are useful for implementing certain features of the preferred embodiment:

- 12 kilobytes of internal programmable read-only memory (PROM);
- 512 bytes of internal read-only memory (RAM);
- 512 bytes of internal electrically erasable programmable read-only memory (EEPROM);
- a serial communication intestate with 32 selectable baud rates:
- a serial peripheral interface:
- an 8-channel, 8-bit analog to digital (A/D) converter; and
- an 8-bit pulse accumulator/generator system.

A detailed description of the M68HC11E9 CPU **901** used in the preferred embodiment may be found in the publication *M68HC11 Reference Manual*, Revision 1, by Motorola, Inc. of Schaumburg, Ill. (1990). CPU **901** is clocked by a 9.8304 MHz crystal which internally divides to a 2.45676 MHz "machine cycle" timing reference for CPU **901** and external devices. CPU **901** is set into a "special bootstrap" mode for updating and testing its internal EEPROM. During the normal operation of the preferred embodiment, CPU **901** is run in its expanded multiplex mode wherein dedicated ports B and C on CPU **901** are converted to 16-bit multiplex address and 8-bit data lines A/D0 through A/D7 and A8 through A15.

In addition to the onboard non-volatile memory and RAM available to CPU **901**, diagnostic base unit **190** further comprises a 32-kilobit by 8-bit static random access memory (SRAM) **902** which is coupled to CPU **901**. Memory **902**, in a preferred embodiment, is a 32-kilobit by 8-bit SRAM, part No. 60L256 manufactured by Motorola, Inc. of Schaumburg, Ill. SRAM **902** is for storing additional information needed by CPU **901**, such as a buffer area for various ports and for use as a scratch pad memory. As is shown in FIG. **9**, pulse accumulator circuitry of CPU **901** is coupled to port pack connector **910** via port A **901***a* to provide communication with various circuitry contained in the peripheral port pack **160** as shown in FIG. **8** via connector **810**. Port A **901***a* doubles as the pulse accumulator/timer circuit and I/O lines for CPU **901**. These lines provide the input capture and output capture compare functions for time signal measurements and generation to and from base unit **190**. Port A **901***a* is used for handling the UUT's input/output (I/O) signals. PAL's **980** provide the necessary address decoding to access SRAM **902**.

Further, port E **901***e* of CPU **901** is coupled to connector **910** which is used as an analog to digital (A/D) port. Port E **901***e* is used for measuring various voltages from the UUT including the battery and the power supply on the UUT. These are measured via channels 1, 2, 3, 4, and 5 of the A/D converter internal to CPU **901** coupled to port E **901***e*. Certain input signals are fed through voltage divider circuits and then through high impedance op amps. The divider and op amp circuit are used to measure voltages up to twice the +5 volt reference provided at V_{RH} such as for the sound connections and certain serial port lines. This increases the gain of certain input signals prior to digitizing by CPU **901**. The low voltage reference V_{LH} for the A/D converter is tied to ground.

Address/data lines **901***c* of CPU **901** are coupled to two port replacement units **920** and **930** which, in the CPU **901**'s expanded mode, allow additional ports to be made available to CPU **901**. When operating CPU **901** in the expanded mode ports B and C of the 68HC11 processor are reconfigured to operate as multiplexed address/data lines A/D0 through A/D7 and A8 through A15. Of course, it will be appreciated by one skilled in the art, that other microcontrollers which do not operate in this manner may not need port replacement units such as **920** and **930**. Port replacement unit A **920** is used for controlling keypad **120** and LCD display **110** of diagnostic base unit **190**. Address/data lines of port **901***c* are further coupled to a second port replacement unit **930**. **930** is coupled to a SCSI interface **940** which is a standard 53C80 SCSI interface integrated circuit manufactured by NCR Corporation for providing SCSI transfers to and from port pack **160** as shown in FIG. **1**.

Address/data lines **903** of CPU **901** are also coupled to ROM pack connectors **950** and **960** which are standard 40-pin CN215J040-G/O female connectors manufactured by Fujitsu Corporation. These provide the coupling between the ROM packs **140** and **150** and CPU **901** for hardware specific diagnostics. Communication with ROM packs **140** and **150** is provided via port replacement unit B **930** and PAL's **970**. PAL's **970** are also coupled to remote port **430** for serial communications to perform remote diagnosis of peripherals or computers over telephone lines via modem.

When CPU **901** is operated in its multiplexed or expanded mode, general purpose I/O ports B and C of CPU **901** configured to operate as multiplexed address-/data lines. Port replacement unit A **920** is coupled to the address/data multiplex lines of port **901***c* on CPU **901** in order to provide communication between keypad **120**, LCD display **110** and unit **190**. Address lines A8 through A11 are processed through a PAL to provide the necessary decoding to generate a chip select signal (/CS) in order to indicate which of the two port replacement units is being accessed. Port replacement unit **920** enables communication between keypad **120**, LCD display **110**, and CPU **901**. Port B **920***b* of port replacement unit **920** is used for communication with LCD display **110**, except for 3 bits of port B which are used for controlling various functions in port pack **160**. The 4 most significant bite of the control information passed through port B are used for controlling display **110**. In the preferred embodiment, display **110** is an LM2433A4C16B dot matrix 4 line by 16 character liquid crystal display module available from Densitron International Corporation. Display **110** requires eight bits of information to generate a character. 4 bits (a nibble) of each datum contained in the control register of port B is written individually to the memory mapped region for LCD display **110** at a time. The high order nibble data is first sent for the display, followed by the low order nibble data in sequential order. When LCD display **110** receives the high nibble of an instruction or display character, it latches the data until it receives the low order nibble data and then the data (character or instruction) becomes available for display.

The remaining four bits of the port B control register **920***b* of port replacement unit A **920** are used for various control functions. Bit **0** (the least significant bit) of port B **920***b* is used to control the command/display mode for LCD display **110**. Bit **1** is used to control the read and write direction of LCD display **110**. The two remaining bits of port B **920***b* are used for controlling the UUT. Bit **2** is used to initiate a "power on" sequence in the Macintosh II brand family of computers available from Apple Computer, Inc. of Cupertino, Calif. via port pack unit **160** as discussed with reference to Table 1 above. Bit **3** of port B **920***b* of port replacement unit A **920** is used to supply power to port pack connector **910** during data transfers as a means of prolonging the battery life of diagnostic tool **100**.

Port C **920***c* of port replacement unit A **920** is used for receiving information from keypad **120**. Keypad **120**, in one embodiment, is a 15-key 9-pin custom silicon rubber keypad. **120** may be one of any number of keypads which are commercially available. The five most significant bits of the keypad memory map register are used for receiving data from keypad **120** in a manner well-known by those skilled in the art. The three least significant bits of the control register are used for enabling-/disabling the columns on the keypad for receiving data input from keypad **120**. Key **921** on keypad **120** (the "*" or "command" key) is used for generating an interrupt request to CPU **901** that a command is being issued through keypad **120**. This is coupled to the maskable interrupt line of CPU **901** to indicate the issuance of a command.

A second port replacement unit B **930** shown in FIG. **9** is used for providing communication with ROM packs **140** and **150**, and port pack **160** in the preferred embodiment. When port replacement B **930** is selected via the chip select signal, it is used for communicating with SCSI interface **940**, and ROM packs **140** and **150**. It is also used for providing control to a UUT serial port **901***d* and for communication over remote port **430**. It can be appreciated by one skilled in the art, however, that any number of computer systems and/or ports may be used depending on the configuration of the port pack, such as **160**, which is coupled to diagnostic base unit **190**. Port C **930***c* of port replacement unit B **930** provides communication with SCSI controller **940**, which is, in turn, coupled to port pack **160** via connector **910**. This provides communication with SCSI devices coupled to a port such as **660** shown in FIG. **6**. Port C **930***c* of port replacement unit B **930** acts as a bidirectional interface to controller chip **940**, but also provides a serial communications port selector line for communication with remote port **430**. In addition, a sleep select pin (/TIRED) is coupled to port C which is used for putting diagnostic apparatus **100** into a mode in which the minimum power is consumed (known as a "sleep" mode). Power consumption is minimized by deactivating the display, and stopping execution of all functions except monitoring keyboard interrupts in a manner well-known to those skilled in the art. SCSI data is memory mapped into RAM area **902** providing bidirectional data transfers between port replacement unit B **930** and SCSI communications chip **940** for reading and writing data between UUT's. Signals for controlling SCSI transfers between communications chip **940** and port replacement unit **930** are well-known to those skilled in the art and are used for SCSI transfers via **940** in the preferred embodiment. One function provided by the preferred embodiment is the use of a software-controllable SCSI termination of the UUT for simulating the presence of device(s) coupled to the UUT. This termination is activated by setting a bit in a control register of port B. Diagnostic base unit **190** can also simulate a SCSI device to a UUT by simulating the presence of partitions, and file allocation tables for performing other types of testing.

Port B **930***b* of port replacement unit B **930** is used as general purpose port for external ROM and RAM page control, as well as ROM pack and serial EEPROM selectors. Address lines A12 through A15 on port **901***c* of CPU **901** are remapped using PAL's **970** to provide the appropriate memory mapping in CPU **901**'s random access memory. Each individual ROM pack is selected via a select bit contained within the control register for the ROM packs. This selects which connector **950** or **960** is accessed for data, to access tests to residing in ROM pack **140** or ROM pack **150**. Each ROM pack such as **140** or **150** may individually contain hardware specific tests.

Diagnostic base unit **190** lastly comprises a serial communication port which is coupled to port D **901***d* of CPU **901**. Three separate serial communication lines are provided through port **901***d* which are activated by two lines on port B **930***b* of port replacement unit **930**. When both of the control bits are cleared, the serial communications port **901***d* is put into a loop back test mode. Either one of these signals being activated enables one of two sets of serial communication transmit and receive lines. Both bits being set enables a third set of transmit and receive lines. Port D **901***d* of CPU **901** is coupled through an RS-232 driver receiver chip (Motorola part No. MC145407-not shown) which is then coupled to port pack connector **910**. The third set of serial transmit and receive lines are used for communication over remote port **430** for remote UUT testing.

The first set of transmit and receive lines are the primary communication means by which the preferred embodiment communicates with software diagnostics embedded in certain types of computer systems, such as the Macintosh brand family of computers. These embedded software diagnostics are collectively known as the "Test Manager." The second set of transmit and receive lines are used for the loop back communication port signals for serial printer ports used in the Macintosh SE and Macintosh II brand family of personal computers.

Standard RS-232 signals are used by diagnostic device **190** for communication with UUT's. However, computer systems such as the Macintosh family of personal computers are capable of using EIA RS-422 hardware protocols which are currently unused by the communications ports directly. The unused lines of the RS-422 lines are selectively sampled using the A/D multiplexers on port pack **160** which, as discussed in more detail below, are connected to port g **901***e* on CPU **901** via port pack connector **910**. A detailed description of one peripheral port pack **160** which may be coupled to diagnostic base unit **190** in one embodiment will now be discussed in more detail with reference to FIG. **10**.

Circuitry of the Port Pack

A block diagram of peripheral port pack **160** which may be used in one embodiment of the present invention is shown and discussed with reference to FIG. **10**. As was discussed with reference to FIG. **6** above, port pack **160** comprises a series of pens **610** through **660** which are coupled via connector **810** to diagnostic base unit **190** in order to test specific types of UUT's. Peripheral port pack **160** has an FCN215Q080-G/O 80-pin male connector **810** manufactured by Fujitsu, Inc. which mates with connector **910** on base unit **190**. A control register of CPU **901** is used to select, via analog multiplexer **1030** shown in FIG. **10**, analog signal channels for measurement. Analog multiplexers **1030** are controlled via analog line selectors **1050** which are coupled to port A **901***a* of CPU **901**. Another signal line coupled to port A **901***a* on CPU **901** is used for enabling SCSI termination. This is coupled via line **1060** from connector **810** to termination circuitry **1010**.

In the preferred embodiment, analog multiplexers **1030** comprise four 74HC4051's available from Motorola, Inc. The selection of each analog MUX **1030** enables any or all of lines **1040** for conversion and measurement of voltage, values by base unit **190**. Also, SCSI and serial data lines **1070** from port pack **160** are coupled to connector **810** for communication with the serial and SCSI circuitry within base unit **190**. An additional feature provided by port pack **160** is UTON select line **1021**. When activated, this line activates power to certain types of UUT's. UTON select line **1021** is coupled to circuitry **1020** to momentarily ground pins **612** and **614** on connectors **610** and **620** in order to initiate a remote power up of the UUT as discussed with reference to Table 1 above. For other types of UUT's not having such a design, the user must activate power manually.

For sampling various voltage levels on connectors **610** and **620** coupled to the UUT, pins **611** and **614** are coupled through analog multiplexers **1030** to connector **810**. In addition, for communicating with the UUT, connectors **610** and **620** are coupled to lines **1070** and then to connector **810**. Communication is thereby provided between the UUT via connectors **610** and **620** and diagnostic base; unit **190** for performing various tasks. Communication via ports **610** and **620** is described in more detail with reference to *Hardware Guide* at pages 287–326.

Serial and SCSI communication is performed with the UUT through connectors **630**, **640**, and **660** is accomplished via the remaining lines **1070** coupled to connector **810**. Sampling of voltages on these connectors is also provided in a similar manner as discussed with regard to connectors **610** and **620** via analog MUX's **1030** across lines **1040**. Sound capabilities of the UUT, in certain systems, may also be tested in this manner. Other analog voltages, as may be appreciated by one skilled in the art, may be sampled in similar manners.

Software Controllable SCSI Bus Termination

A detailed description of the software controllable SCSI termination circuitry **1010** of the preferred embodiment will now be discussed with reference to FIG. **11**. The lines coupled from connector **810** to SCSI connector **660** are routed through **1010** for providing the bus termination. Five volts is received on line **1121** from pin **685** on connector **660** to circuitry **1010**. One signal line, REG3 **1060** from base unit **190**, is provided which activates termination. The remaining SCSI lines **1100** from connector **660** are coupled to the 110 ohm resistors of resistor packs **1110**, and resistor pack **1110** is coupled to the cathode on diodes in either of diode packs **1111**, **1112**, or **1113**. The anodes of all the diodes in diode packs **1111**, **1112** and **1113** are tied together and connected to an output line **1114** of the low drop out positive adjustable voltage regulator **1115**. Voltage from **1117** is fixed by resistors **1122** (**121**ω)and **1123** (**249**ω) which are coupled to ground on **1128** on **1115** and the output **1114** through a series of capacitors (**1126** and **1172** and **1124**) tied to ground **1125**. The input **1116** of device **1115** is coupled to the drain **1118** of a dual P-channel enhancement MOS-FET **1117**. The source **1120** of **1117** is coupled to the termination power line **1121** (which carries +5 volts) of the UUT (pin **685** on connector **660**). Gate **1119** of MOS-FET **1117** is coupled to signal line **1060** coupled to a register in base unit **190** received from for activating SCSI bus termination. When this bit is set to a logic 0, a negative voltage is created at gate **1119** of **1117** effectively sourcing +5 volts From pin **685** (termination power) to **1116** on **1115**. **1115** steps down the voltage to 3.8 volts which is supplied to diode packs **1111**, **1112**, and **1113**. Diode packs **1111**, **1112**, and **1113** drop the voltage down another one volt to provide 2.8 volts to resistor pack **1110**. The 2.8 volts with 250 mA limiting resistors is provided to the remaining SCSI lines **1100** and meets the proper voltage and current specifications for SCSI termination of the bus.

On the other hand, when a logic "1" is present on signal line **1060**, the voltage at gate **1119** of **1117** is equalized, and thus removing SCSI bus termination from SCSI signal lines **1100**. No voltage is provided from drain **1118** of device **1117** to **1116**. Each of the resistors in resistor pack **1110** appears as an individual open circuit to SCSI connector **660**. SCSI termination is thus effectively removed. In summary, an active signal over line **1060** causes **1010** to deactivate SCSI bus termination on lines **1100**, and a logic 0 received over line **1060** causes **1010** to activate SCSI bus termination on lines **1100**.

Testing UUT's

The testing of UUT's coupled to diagnostic tool **100** via port pack **160** in the preferred embodiment is accomplished via a sequence of routines embedded in ROM packs **140** and **150** which perform certain tests embedded in those ROM packs, or causes accesses to built-in diagnostics contained within the UUT. In the Macintosh family of personal computers, a series of routines known as the "Test Manager" is available in the system ROM of each computer. On a system which is not even able to power up and complete bootstrap initialization, this program will be unavailable. Therefore, certain "low-level" tests perform certain basic diagnostics such as testing a "power on" battery on the motherboard, testing voltage levels on certain signal lines, determining whether there is SCSI termination, or measuring the voltage of SCSI termination. A summary of these tests is set forth below. Each of these low level tests is written in 68HC11 assembly language for CPU **901** in the preferred embodiment but may be written in high level languages such as the "C" or Pascal programming language in alternative embodiments. These tests are performed without entry into the diagnostic program by measuring voltages available on the various connectors. etc. which are coupled to tool **100** and a UUT. These tests are as follows:

Tests that Do Not Require the Test Manager to Function

Power Supply (PowrS)

This text measures the voltage at the power supply of the UUT. The value can range from 0.0 volts to 5.5 volts. This test runs on UUT's in the Macintosh brand of personal computers. This test has no pass/fail. It displays the voltage with a message stating what range of values indicates a functional power supply (usually >4.5 volts). This test requires a cable to be connected to a UUT and either port **610** or **620**.

Battery Voltage (Batt.)

This test measure the battery voltage off the ADB cable. The value can range from 0.0 volts to 10.0 volts. This test runs on Macintosh II motherboard and IIx brand computers. This test has no pass/fail. It displays the voltage of the battery with a message stating what range values indicates a functional power supply (usually >6.5 volts). This test requires a cable to be connected from a UUT to either port **610** or **620**.

Power Up Voltage (PwUpV)

The Macintosh IIcx brand computer uses a trickle current from the power supply to provide power up voltage (which is provided by a battery on the motherboard). This test measures the trickle current off a cable coupled to **610** or **620**. The value can range from 0.0 volts to 10.0 volts. It is very similar to battery voltage. This text has not pass/fail. It displays the voltage with a message stating what range of values indicates a functional power supply (usually >4.5 volts). This test requires a cable to be coupled to a UUT and either port **610** or **620**.

Power On CPU

This function, by imitating the action of pressing the soft power on key activates power in the UUT. This function runs on Macintosh II, IIx and IIcx brand personal computers. This function has no pass/fail. This test requires one cable to be connected from a UUT to port **610** or **620**.

SCSI Termination Check

This test checks the SCSI lines to see if a good termination exists on the bus. This test runs on all Macintosh brand personal computers. This test has pass/fail only. This test requires a SCSI cable to be connected from port **660** to the UUT.

SCSI Termination Power

This test measures the termination voltage off the SCSI bus. The value can range from 0.0 volts to 5.0 volts. This test runs on all members of the Macintosh brand personal computer family although may also run on other computers having a SCSI connector similar to **600**. This test has no pass/fail. It displays the termination voltage with a message stating what range of values indicates a functional terminator (usually >4.5 volts). This test requires a SCSI cable to be connected from connector **660** to the UUT.

Termination [On/Off]

This function turns on (or off depending on the previous state) the internal termination power of the SCSI bus in diagnostic tool **100**. This function runs on all members of the Macintosh brand personal computer family. This function has no pass/fail. It displays the termination condition as on or off on the screen. This function requires the SCSI cable to be connected to connector **660** and the UUT.

SCSI Reset

This function forces a hard SCSI reset on the bus. This function runs on all SCSI type hard drives. This function has no pass/fail. This function requires a SCSI cable to be connected to connector **660** and the drive.

SCSI Bus Scan

This test scans the SCSI bus looking for hard drives. It builds a table of any drives it finds and allows the user to obtain information about each drive such as drive type manufacturer and size. This test runs on all SCSI type hard drives. This test has no pass/fail. This test requires the SCSI cable to be connected to connector **660** and the SCSI bus being tested.

Test Manager Entry ("TstMd")

This function enters the Test Manager on the UUT. It cycles power and uses the SCSI bus to jump into the Test Manager on the UUT. This function runs on all machines equipped with a SCSI chip and having a Test Manager diagnostic program. This function fails only if it is not successful in entering the Test Manager. This function requires that cables be connected from the UUT to **660**, **610** or **620**, and a serial cable for the modem to be connected to port **630** or **640**. The process of entering the Test Manager is discussed in more detail with reference to FIGS. 12*a* through 12*d*.

Entering the Test Manager Diagnostic System

If the system is able to receive system power, however, the Test Manager diagnostic program may be entered in the UUT to perform certain tests at the user's choosing using diagnostic tool **100**. There are two ways in which the Test Manager is entered in a Macintosh brand personal computer using the preferred embodi-

ment: by manually initiating a "non-maskable interrupt" (NMI) switch 620 on a Macintosh brand personal computer; or by causing the UUT to enter the Test Manager by simulating a SCSI device such as a disk drive and jumping to the Test Manager. The former option is difficult because the NMI switch must be issued within five to ten seconds after the UUT commences a bootup. Fairly tight timing constraints must be adhered to in order to enter the Test Manager in this manner. In UUT's which are able to initiate system power and perform bootstrap initialization from an attached SCSI device, the latter method to enter the Test Manager is preferred. This is discussed in more detail with reference to FIGS. 12 through 15.

FIGS. 12*a* through 12*d* show one process 1200 which checks certain basic conditions in the UUT and attempts to enter the Test Manager. Prior to bootstrap initialization from the simulated SCSI device provided by device 100, certain basic conditions need to be established and tested for. Various corrective measures are taken to try to enter the Test Manager if entry is not successful on the first attempt. This process is shown in FIGS. 12*a* through 12*d*. When a UUT is connected to tool 100 for performing diagnostics, process 1200 is executed by tool 100 in order to attempt to enter the Test Manager. Process 1200 starts at step 1201, as shown in FIG. 12*a*, by displaying to the user at step 1202 that power-up routine is being initiated in the UUT. At step 1203, the SCSI bus termination and boot attempt counters, which are used for multiple attempts to terminate the SCSI bus or attempt to initialize the system, are cleared. The serial modem port 430 on tool 100 is initialized at step 1204 in order to prepare the unit for communication with a UUT. At step 1205, it is determined whether the UUT is already powered up. If power is present in the UUT, then process 1200 proceeds to step 1221 as shown in FIG. 12*b*. It is determined whether the UUT is powered up, in a Macintosh brand computer system, using techniques well-known to those skilled in the art by sampling certain lines coupled to either ports 610 or 620 of peripheral port pack 160. If the UUT is not powered up, as determined at step 1205, then process 1200 proceeds to step 1206 wherein a power-up is attempted on the UUT using the sequence issued over port 610 or 620 on port pack 160 for certain models of the Macintosh brand computer. Again, it is determined at step 1207 whether the UUT has system power, and if it does, then process 1200 proceeds to step 1212 on FIG. 12*b*. If not, as determined at step 1207, the message "Waiting for power on" is displayed at step 1208, and process 1200 in FIG. 12*a* proceeds. It is then determined, at step 1209, whether the "Back" key has been depressed by the user. This key allows the user to abort from any process currently executing in tool 100. This is used if the user wishes to escape out of the waiting for power on loop of steps 1209 and 1210. If the "back" key is depressed, then process 1200 returns from Test Manager entry sequence 1200 at step 1211. If the "back" key has not been depressed as determined at step 1209, then the UUT is tested again at step 1210 to determine whether system power is present. Steps 1209 and 1210 are repetitively performed until it is detected that the UUT has powered up as determined at step 1210. Once power is present, step 1210 proceeds to step 1212 in FIG. 12*b*.

As shown in FIG. 12*b*, process 1200 continues at step 1212 wherein it is determined whether SCSI bus termination is present in the UUT. If SCSI bus termination is present, then 1212 branches to step 1218 to continue the initialization. If SCSI bus termination is not present, then process 1200 proceeds to step 1213. 1213 determines whether this is the second attempt to terminate the SCSI bus, and if it is, step 1213 proceeds to step 1214 wherein the message "Cannot terminate SCSI (cable problem?), Try manual power on" is displayed to the user. Then, process 1200 branches to the exit process at step 1228 shown in FIG. 12*d*, and process 1200 is returned from at step 1235. If, however, this is not the second attempt to terminate the SCSI bus, then step 1213 proceeds to step 1215 wherein the user is prompted with the message that "SCSI termination is missing," and the user is given the option to either "1) Terminate the bus or 2) cancel" the present operation. It is determined at step 1216 which option the user selected. If "cancel" is selected, step 1216 proceeds to step 1235 on FIG. 12*d* and process 1200 is returned from. If, however, the "cancel" selection was not made, as determined at step 1216, then SCSI termination is attempted to be activated at step 1217 as shown in FIG. 12*b*. SCSI termination is performed, in the manner as discussed earlier, using soft SCSI termination circuitry 1010 as shown in FIG. 11. Then, the message "Waiting for SCSI block read" is displayed at step 1218, wherein tool 100 waits until the UUT attempts to read from the simulated SCSI device over port 660. Tool 100 will cause the UUT to enter fine Test Manager by "feeding" blocks to the UUT at step 1219. This is discussed in more detail with reference to FIG. 13. Tool 100 then waits three seconds in order for the Test Manager to be entered at step 1220 after the issuance of the entry sequence at step 1219. It is determined at step 1221, whether the UUT is in the Test Manager. If the UUT is not in the Test Manager, then process 1200 proceeds to step 1222 in FIG. 12*c*. If, however, the UUT was in the Test Manager, then process 1200 proceeds to step 1224 in FIG. 12*d*.

As shown in FIG. 12*c*, process 1200 continues at step 1222 wherein it is determined whether UUT power was on already, or whether the power-on sequence at step 1206 had to be performed. If power was already on in the UUT, then process 1200 proceeds to step 1229 wherein it is ascertained whether this is the second attempt to boot the Test Manager in the UUT. If it is the second attempt to boot, then the message "SCSI/serial problem. Try manual entry (press BACK)" is displayed at step 1223. If process 1200 activated power in the UUT, then step 1222 also proceeds to step 1223 to display the message. After the message if step 1223 is displayed, the exit process at step 1228 shown in FIG. 12*d* is performed, and process 1200 is returned from at step 1235. If this is not the second boot attempt as determined at step 1229, then the message "Waiting for power off" is displayed at step 1230, and the UUT is checked via ports 610 and 620 to see whether power is off in the UUT at step 1231. If it is not, and the "back" key was not depressed (causing interruption of process 1200), then steps 1231 and 1232 are performed repetitively until it is determined at step 1231 that power is off in the system, or the "back" key is depressed. When power-off is detected in the UUT at step 1231, then process 1200 proceeds back to step 1206 to issue a "Power-on" signal to the UUT as shown in FIG. 12*a*. If the "back" key is depressed, as determined at step 1232 (interrupting Test Manager Entry Process 1200), then process 1200 is returned from at step 1235 shown in FIG. 12*d*.

If the UUT was not in the Test Manager as determined at step **1221** in FIG. **12***b*, then process **1200** proceeds to step **1224** in FIG. **12***d*. Step **1224** in FIG. **12***d* checks the UUT's timing. This is determined using techniques well-known to those skilled in the art by sampling various signals received over the ports on port pack connector **160** to see if the UUT is responding within defined tolerances. If the timing is bad as determined at step **1225**, then process **1200** proceeds to step **1226**. If the timing is not bad as determined at step **1225**, then the message "Communications established: CPU is in Test Manager mode" is displayed at step **1233**, and tool **100** waits an additional five seconds for a user intervening keypress, at step **1234**. If no keypress is received, process **1200** returns at step **1235**. If, however, the timing was bad as determined at step **1225**, then process **1200** proceeds to **1226** to determine whether tool **100** activated power in the UUT. If power was on already (tool **100** didn't activate the power), then a "RESET" command is issued to the UUT at step **1236** and process **1200** continues at step **1221** as shown in FIG. **12***b*. If power was activated in the UUT by tool **100**, then the message "Test Manager OK but timing problem found" is displayed at step **1227**, and the exit process is branched to at step **1228**. Then, process **1200** is returned from at step **1235**.

Thus, at the end of process **1200**, the UUT should be in its Test Manager mode, or the user is prompted with various options that he may take in order to perform diagnostics on the UUT. The entry into the Test Manager by simulating a SCSI device mentioned with reference to step **1219** will now be discussed.

Entry Into the Test Manager by Simulating a SCSI Device

Entry into the Test Manager is performed by diagnostic tool **100** by simulating a SCSI device to the UUT. This is accomplished by sensing signals received through connector **660** transmitted by the UUT. When the appropriate SCSI initialization signals are received through port pack **160** from a UUT coupled to connector **660**, diagnostic tool **100** issues response signals through connector **660** to the UUT to cause the system to jump to the Test Manager. It performs this by simulating a SCSI device such as a disk drive using a driver descriptor map (DDM) and partition map entry which is expected by certain UUT's using SCSI devices. These entries are shown in more detail in FIGS. **14** and **15**. As is well-known to those skilled in the art, a computer system having a SCSI interface, such as a UUT probes the SCSI bus to determine if SCSI devices are present. Each SCSI device identifies itself with an identification number. This is known as the arbitration phase. Once ID's have been sampled by the UUT, selection of a device may be accomplished. This is known as the selection phase. After arbitration and selection, the UUT can issue a command (such as a read or a write) and the device will respond with data. The preferred embodiment senses the issuance of the arbitration and selection signals, and simulates a SCSI device with an ID=6; which is the default highest priority SCSI ID number besides the UUT in a Macintosh brand computer. Other responses to probes by the UUT are now discussed. These are performed, as is well-known to those skilled in the art, by sensing the arbitration, selection, and command signals sent by the UUT, and device **100** responds in a manner expected from a SCSI device. The signals issued to the UUT will now be discussed.

The SCSI Test Manager entry process performed by the UUT is shown as **1300** in FIG. **13** (referred to in step **1219** of process **1200**). Process **1300** is performed by a UUT when allowed to continue to perform bootstrap initialization and has a SCSI device coupled to one of its ports through port **660**. Process **1300** starts at step **1301** wherein, at step **1302**, a first available SCSI device on the bus is selected that has an ID=6 (this is the highest priority SCSI device in a Macintosh brand computer). When the UUT requests an access to the SCSI device having ID=6, diagnostic base unit **190** senses this signal and transmits responsive data to the UUT via port pack **160**. The format of data responsive to requests issued by the UUT is shown in FIG. **14** and is shown in more detail in FIGS. **14** and **15**. At step **1303**, the UUT will attempt to read the first 256 bytes of block 0 of this simulated device by issuing read commands through port **660** to tool **100**. Block 0 **1401** is shown with reference to FIG. **14**. This contains the driver descriptor map (DDM) for the simulated SCSI partition. The UUT attempts, at step **1304**, to look for a driver descriptor map such as **1401** with a value equaling $4552 (this indicates that the SCSI device has been formatted). This is indicated as field **1501** in FIG. **15**. Upon receiving the address for the first 256 bytes of block 0 of the unit with ID=6, diagnostic tool **100** transmits DDM **1401** onto the SCSI bus. Driver descriptor map **1401** comprises two portions, **1401***a* and **1401***b*. **1401** a contains "real data" which resides in non-volatile memory of tool **100** and is transmitted to the requesting UUT. The remaining portion **1401***b* of the data is zero-filled and transmitted to the UUT through port **660** byte-by-byte from a register in tool **100**. The remaining blocks shown in FIG. **14**: **1402***b*, **1403***b*, **1404***b*, and **1405***b* are similarly zero-filled and transmitted to the UUT. The remaining "real " portions of driver descriptor map **1401***a* is shown in FIG. **15**. Driver descriptor map **1401***a* indicates the size in blocks of the device in field **1502** (a one word field) and the number of blocks on the device in field **1503** (a 32-bit longword). Blocks are 512 bytes long, in a preferred embodiment. Field **1504** indicates the device type (a one for disk drive), **1505** has the device ID type (one). Field **1506** also contains a one, in the preferred embodiment, a value expected by Macintosh brand UUT's. The number of driver descriptors is contained in field **1507**. In this example, there is only one. The driver descriptor starts at field **1508**. The first block of the driver resides at block four as indicated by field **1508** (a 32-bit longword), and the driver size is exactly one block long as indicated by field **1509**. Lastly, the driver type is of type **1**, which is contained in field **1510**, in the Macintosh brand family of personal computers. After the driver descriptor map is read at step **1304**, step **1305** reads the pseudo device driver of the simulated SCSI device at block **4**, shown as **1405** in FIG. **14**. The format of blocks **1402** through **1405** are discussed with reference to FIG. **16**.

Partition map entries, such as **1402**, **1403**, **1404**, or **1405** are one-block entries containing information about the SCSI partitions which are being accessed. The pseudo device driver of the preferred embodiment resides in one of these partition map entries, and is shown as **1405** in FIG. **14**. Each of the partition map entries has a specific format, which is well-known to those skilled in the art, and is set forth in *Inside Macintosh*, Volume V available from Addison-Wesley at pages V-576 through V-582. Because the actual data contained within fields in blocks such as **1402** through **1405** is well-known to

those skilled in the art, a detailed discussion of that will not be set forth here. Each of the blocks **1402** through **1405** contains a first portion such as **1402***a* through **1405***a*, which contains real data in order to conserve space in the ROM of the preferred embodiment. The remaining portions such as **1402***b* through **1405***b* contain dummy data which is transferred to the UUT when request for the remainder of the block is made by the UUT. In each of the partition map entries **1402** through **1405**, the actual portion read in by the UUT, **1402***a* through **1405***a*, is 64 bytes in length. The remaining portions **1402***b* through **1405***b* are padded with zeros. The partition map entries are then read by the UUT. At step **1306**, the partition map **1402** for the operating system is read. Once this is done, the partition map entry for the pseudo partition is read at step **1307**. Then, the partition map entry **1404** for the driver is read in at step **1308**. This gives the system the address at which to start to execute the driver contained in **1405**. Then, at step **1309**, the partition map entry for the operating system **1402** is read again. At step **1310**, the driver residing at the location indicated by **1404** is called to install itself and the driver causes a jump to the Test Manager to occur. Once process **1300** is complete, at step **1311**, the Test Manager is running in the UUT, which can now be accessed and communicated with by diagnostic tool **100**.

In summary, diagnostic tool **100**, in one embodiment of the invention, simulates a SCSI device with an ID=6 for a Macintosh brand personal computer. A series of requests is made to the device to read various information off a simulated SCSI device. Blocks are returned from this simulated device as shown in FIG. **14** and described with reference to FIG. **13**, and the diagnostic device **100** returns blocks which are expected by the UUT. Blocks are fed to the UUT in the following order: **1401**, **1405**, **1402**, **1403**, and **1402**. As a last step, the device driver residing in block **1405** is called to execute itself thus causing a jump to occur to the Test Manager which resides in the UUT system ROM.

Once entry into the Test Manager of the UUT has been performed, various tests within the Test Manager may be accessed and executed. It can be appreciated by one skilled in the art that other architectures possessing similar test routines may be accessed in a similar manner. This will allow entry and testing in an automated fashion by diagnostic apparatus **100**. The commands set forth in Table 4 are provided for execution of the Test Manager remotely via a coupling with the UUT to serial port **640** on port pack **160**. Each command is preceded by a "*" which indicates that a command is following on serial port **640**. When a UUT is activated and the Test Manager is entered, in one embodiment, the serial port of the UUT defaults to a 9600 baud configuration and expects "*" commands to be received via serial port **640** coupled to the serial port of the UUT. Responses to tests requested by diagnostic apparatus **100** are also provided across the serial port. Specific commands are summarized as follows:

TABLE 4

Initiating Command	UUT Response	Command Name	Description
*S	*S	Stop initial input message and Start UUT ROM Monitor	This command is the "handshake" that initializes the UUT ROM monitor to accept test commands. It also silences the UUT (if the UUT sends out a repeating error message upon failure or if the ROM monitor has been invoked).
*L xx xx xx xx	*L	Load address	This command is followed by 4 bytes that specify the address where the UUT will start loading (or downloading) data into RAM. Response indicates that the UUT received the valid command (handshake). Can be used to load tests into the UUT.
*B xx xx	*B	Byte count	This command is followed by 2 bytes that specify the number of bytes to be downloaded. (*L must precede this command). Response indicates UUT received the valid command (handshake).
*D xx	*D	Download data	This command is followed by xx xx (2 hex) bytes as specified by the *B command. (*L & *B must precede this command). Response indicates UUT received valid command (handshake).
*C xx xx xx xx	*C	Checksum data	This command requests memory range checksum specified by starting location and number of bytes (via *L and *B respectively). Returns checksum followed by *C. Must be used with *L and *B.
*G xx xx xx xx	*G	Go . . . (execute)	This command is followed by 4 hex bytes that specify the address where the UUT will start running a program. (To run a downloaded program, *L, *B, and *D must precede this command). Response indicates UUT received the valid command (handshake).
*O xx xx xx xx	*O	O = low or bottom RAM test address	This command is followed by 4 hex bytes that specify the address where the UUT will start running a RAM test. (A pointer). Response indicates UUT received the valid command (handshake).
*1 xx xx	*1	1 = high or top	This command is followed by 4 hex bytes that specify the address where the UUT will start running a RAM test. (A pointer). Response indicates UUT

TABLE 4-continued

Initiating Command	UUT Response	Command Name	Description
			received the valid command (handshake).
*2 rr	*2	Read CPU register	Requests CPU register data specified by rr. Response returns 32-bit register information from the UUT CPU register, rr. The register dump is followed by *2 to indicate the end of transmission.
*3 rr xx xx xx xx	*3	Write CPU register Registers: D0-D7 SP A0-A6 SR	Writes 32-bit value to register specified by rr. Example: *3D301F9AB35*3 *2SP000E952*2 Underlined section returned by Test Manager)
*4	*4	Clear UUT error registers. (e.g., D6.L & D7.W)	Clears the error storage registers on the unit under test. Response indicates UUT received the valid command (handshake).
*5	*5	Re-enter ROM monitor	Re-starts ROM monitor entry and re-calls generic header (e.g., *APPLE*xxxxxxxxxxx*1*). Response indicates UUT received the valid command (handshake).
*A	*A	ASCII mode	Sets ASCII character mode. Sends and receives ASCII data. Response indicates UUT received the valid command (handshake).
*H	*H	Hex mode	Sets hex character mode. Sends and receives hex data. Response indicates UUt receives the valid command (handshake).
*R xx xx xx xx xx xx	*R	Result request	Requests current status or test results. Response returns a 12-character result code that specifies the current error status.
*M xx . . .	*M	Memory dump	Requests memory data specified by *L and *B (*L and *B must precede this command). Response returns memory information as specified by the *L and *B commands. The memory dump is followed by *M to indicate the end of transmission.
*E	*E	Echo	Requests the UUT to echo externally transmitted characters back to the external device. Response indicates UUT received the valid command (handshake).
*I	*I	Initialize	Requests the UUT to re-initialize itself. This terminates serial communications. (Re-boot).
*T tt tt pp pp oo oo	*T	ROM-resident Test request	Requests the UUT ROM test to be run. This command is followed by 8 hex bytes; "tt tt" is the "pass count" (or number of times test is to be run); "oo oo oo oo" is the option word. Response indicates UUT received the valid command (handshake). Examples of ROM Test number (tt tt) activities: 0000 - Size Memory 0001 - DataBus Test 0002 - Mod3Test 0003 - Addrline Test 0004 - ROMTest 0005 - Rev3ModTest 0006 - StartUpROMTest

An additional feature provided by the Test Manager is the ability to download specific tests into the Test Manager. This is performed using the "*D" command and such tests may be executed by typing in the "*G" command with the address of the test loaded using the "*D" command. The last command in any user-specified test using "*D" must jump back to the starting address of the Test Manager to provide the appropriate response to diagnostic tool **100**. It can be appreciated by one skilled in the art that a variety of entries into system-specific test functions such as the ones provided above may be performed.

Tests Requiring the Test Manager

Most of the tests run on diagnostic tool **100** require the UUT to be in the Test Manager. This requires the machine to have a limited amount of functionality. Most of these tests download a piece of code into the UUT to run in the native environment. If the user attempts to run one of these tests without being, in the Test Manager, he will be given a prompt to enter the Test Manager by diagnostic tool **100**. In order to enter the Test Manager automatically, each test requires that the UUT be coupled to ports **660** and **610** or **620** using appropri-

ate cables. Also, tool **100** must be coupled to the UUT through port **630** using a serial cable to communicate with the Test Manager.

ROM Checksum (ROMck)

This test checksums all of the ROM's in the CPU (if more than one exists). This test will not identify individual ROM failures. This test fails if the checksum does not match the one stored in the ROM, otherwise it passes.

RAM Size (RAMsz)

This test determines the size of the RAM in the CPU. The test finds the largest SIMM in the bank and uses that to determine the size. For example, if the test finds three 256K SIMM's and one 1 Meg SIMM, it will size the machine to 4 Megs. This test has no pass or fail. It reports the size of memory it finds and indicates if it thinks the memory is misconfigured (a system where the largest SIMM's are in bank A instead on bank B) or bad (which may just be mismatched (a system which has different size SIMM's in the same bank)).

Address Test (Addr.)

This test checks the address lines of the CPU. It determines if the lines are functioning properly. This test has pass or fail only.

Databus Test (Data)

This test checks the data lines of the CPU. It determines if the lines are functioning properly. This test graphically shows catastrophic errors and missing SIMM's.

SIMM's Test (SIMMs)

This test uses the results from RAM Size to read and write RAM to determine if it is good. This test graphically shows catastrophic errors, missing SIMM's and individual bad SIMM's.

VIA Test (VIA)

This test checks the VIA's (Versatile Interface Adaptors or Peripheral Controller Chips) for functionality. This test gives pass or fail only.

Clock Test (Clock)

This test checks the UUT real time clock chip for functionality. This test gives pass or fail only.

Parameter RAM Test (P/RAM)

This test verifies that the PRAM (parameter RAM which controls data, and mouse information) can be read from and written to. This test gives pass or fail only. NOTE: There are two modes of operation for this test. One mode saves and restores the contents of PRAM (parameter RAM of the UUT), and the other mode clears out all of PRAM.

SCC Test (SCC)

This test verifies that the SCC chip functions in all modes and that the serial bus can send and receive data correctly. This test gives pass or fail only.

SCSI Test (SCSI)

This test verifies that the SCSI (small computer system interface) chip functions in all modes and that the SCSI bus can send and receive data correctly. This test gives pass or fail only.

SWIM/IWM Test (SWIM)

This test determines if the CPU has an IWM or a SWIM (floppy disk controller chips) connected and then tries to initialize the chip. This test gives pass or fail only. This test can only detect catastrophic failures in the IWM/SWIM. The floppy drive tests provide a more comprehensive test of the chip.

FPU Test (FPU)

This test verifies that the FPU chip (floating point math co-processor) functions. This test runs on Macintosh II, IIx, IIcx brand family of personal computers. This test gives pass or fail only.

Sound Test (Sound)

This test verifies that the sound chip registers function and that data can be read from and written to the chip. This test also measure the sound out for volume and frequency. This test gives pass or fail only.

ADB Test (ADB)

This test verifies that the ADB (mouse and keyboard) transceivers, the portions of the VIA that control ADB, the ADB line and the ADB port is functional. This test gives pass or fail only. This test requires that both ADB cables be connected from the UUT to ports **610** and **620**.

Video Card Test (Video)

This test determines what video cards are in what slots and builds a menu for the user. The user can select which card he wants to test. The test will then test the RAM on the selected video card. This test runs on the Macintosh II, IIx, and IIcx brand family of personal computers. This test gives graphic representation of which SIMM's are bad or fails the card if the soldered SIMM's are defective.

Floppy Drive Test (Drive)

This test determines what floppy drives are in what ports and builds a menu for the user. The user can select which drive he wants to test. It then tests the ability of the drive to read, write, and seek. This test gives pass or fail only.

Power Off CPU

This test shuts off power to the CPU. This test funs on Macintosh II, IIx, and IIcx brand family of personal computers.

Thus, an invention for diagnosis for computer systems has been described. Although the present invention has been described particularly with reference to FIGS. **1** through **15**, it will be apparent to one skilled in the art, however, that the present invention has utility far exceeding that disclosed in the figures. It is contemplated that many changes and modifications may be made, by one of ordinary skill in the art, without departing from the spirit and scope of the present invention as disclosed above.

What is claimed is:

1. An apparatus for testing computer systems comprising:

a. a base unit, the base unit comprising a central processing unit for executing instructions, issuing commands, and receiving data from a first computer system;

b. a first connecting module which contains at least one connector for coupling to the first computer

system, said connector being coupled to an existing connector on the first computer system, said first connecting module being interchangeable with a second module for interfacing with a second computer system, said at least one connector comprising a disk drive connector; and

c. a first nonvolatile memory module coupled to the base unit, the first nonvolatile memory module comprising a first set of tests for the first computer system, the first nonvolatile memory module being interchangeable with a second nonvolatile memory module comprising a second set of tests for a second computer system, said first set of tests including an initialization circuit causing said first computer system to initially attempt to execute an internally-stored set of diagnostic routines, and if said initialization circuit is unsuccessful at causing said first computer system to attempt to execute said internally-stored set of diagnostic routines, then said initialization circuit causing said apparatus to emulate a disk drive through said disk drive connector and causing said first computer system to perform a bootstrap initialization of said first computer system from said apparatus by loading a set of externally-stored diagnostic routines from said first nonvolatile memory module.

2. The apparatus of claim 1 wherein the base unit comprises a keyboard for entering commands and a display for displaying information to a user.

3. The apparatus of claim 1 wherein the base unit comprises a connector which may be coupled to a third computer system which remotely controls the base unit and issues commands through the base unit to the first computer system.

* * * * *